

Real-Time Network Management Final Scientific and Technical Report

Submitted to:

**Commander
U.S. Army Aviation and Missile Command
Attn: AMSAM-RD-WS-DP-SB (Mr. Roach, Technical Monitor)
Building 7804, Room 223
Redstone Arsenal, AL 35898-5248**

In reference to:

Contract No. DAAH01-99-C-R129

Submitted by:

**BAE SYSTEMS Portal Solutions Inc.
(formerly Synectics Corporation)
10400 Eaton Place, Suite 200
Fairfax, VA 22030**

**111 East Chestnut Street
Rome, NY 13440**

November 2001

**DISTRIBUTION STATEMENT A
Approved for Public Release
Distribution Unlimited**

BAE SYSTEMS

20011128 069

**REAL-TIME NETWORK
MANAGEMENT**

**Final Scientific And
Technical Report**

NOVEMBER 2001

Sponsored by

Defense Advanced Research Projects Agency
ITO

Issued by U.S. Army Aviation and Missile Command Under

Contract No. DAAH01-99-C-R129

DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited.

REPORT DOCUMENTATION PAGEForm Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave Blank)		2. REPORT DATE November 2001	3. REPORT TYPE AND DATES COVERED Final Technical Report, April 1999 – November 2001
4. TITLE AND SUBTITLE Scientific and Technical Report, Final Technical Report			5. FUNDING NUMBERS C - DAAH01-99-C-R129
6. AUTHOR(S) Joseph J. Riolo			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) BAE SYSTEMS Portal Solutions Inc. (formerly Synectics Corporation) 111 East Chestnut Street Rome, NY 13440			8. PERFORMING ORGANIZATION REPORT NUMBER 99902-000-A004
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) U.S. Army Aviation & Missile Command AMSAM-AC-RD-AY Redstone Arsenal, AL 35898			10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES			
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.			12b. DISTRIBUTION CODE
13. ABSTRACT (Maximum 200 words) The objective of this effort was to develop methods for diagnosing network performance problems in real time. We broke the program down into four tasks. The first task entailed the research and categorization of models that provided both a sufficiently accurate description of the behavior of the computer network and were computationally tractable for incorporation into a real-time network monitoring system. Auto Regressive models were used to easily model the traffic patterns as well as predict the future traffic pattern on the nodes. Secondly, we researched mechanisms for achieving the real-time determination of the parameters of the current-to-be models of a network and for monitoring k-steps ahead forecasting accuracy of the current models. Cumulative Sum method was used to identify a model change point in the traffic pattern. This method looked at the mean and variance of the traffic and provided information on how much the traffic had changed and in what way. Thirdly, we built an object model of the analytical and operational entities that made up the architecture. Finally, we developed the application based on the research and design of the previous tasks using Java, with support from Java Management Extensions and Java database Connectivity APIs.			
14. SUBJECT TERMS network management, network traffic, SNMP, auto regressive models, event correlation, cumulative summation filters, data packets, Java, JMAPI, JDMK			15. NUMBER OF PAGES 74
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

REAL-TIME NETWORK MANAGEMENT

BAE SYSTEMS
Portal Solutions Inc.
111 East Chestnut Street
Rome, New York 13440

Mr. Joseph J. Riolo, Principal Investigator
Telephone: 315-337-3510

Effective Date of Contract:
14 April 1999

Final Scientific and Technical Report

Contract Expiration Date:
12 October 2001

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either express or implied, of the Defense Advanced Research Projects Agency of the U.S. Government.

DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited.

TABLE OF CONTENTS

1.0	INTRODUCTION-----	1
2.0	OBJECTIVES-----	1
2.1	Task 1 – Mathematical Modeling-----	2
2.1.1	Network Traffic Model Research-----	2
2.1.2	Model Change Point Research-----	5
2.1.3	Queuing Model Research-----	7
2.1.4	Aggregate Model Research-----	11
2.1.4.1	Algorithm 1-----	13
2.1.4.2	Algorithm 2-----	14
2.1.4.3	Algorithm 3-----	14
2.1.4.4	Algorithm 4-----	14
2.1.4.5	Algorithm 5-----	16
2.2	Task 2 – Real-Time Model Selection-----	16
2.2.1	Real Time Estimation of Model Parameters-----	16
2.2.2	Model Comparison Research-----	19
2.2.3	Model Change Point Identification-----	21
2.3	Task 3 – Object Modeling and Software Design-----	29
2.3.1	Object Modeling our Software Prototype-----	29
2.3.2	Graphical User Interface-----	34
2.3.3	SNMP Network Management with JDMK-----	37
2.3.4	Model Based Diagnostics-----	40
2.3.5	Future Diagnostics Input-----	43
2.3.6	Study of Mobile Agents-----	45
2.3.7	Network Management Load Distribution-----	47
2.3.7.1	T (Transitional)-----	48
2.3.7.2	QT (Quasi-transitional)-----	48
2.3.8	Scalability Issues of Traffic Models-----	49
2.3.9	Alternative Frameworks-----	49
2.4	Task 4 – Software System Implementation-----	50
2.4.1	Software Evaluations-----	50
2.4.2	SNMP Data Collection Implementation-----	52
2.4.3	CUSUM and AR Design-----	53
2.4.4	The Implementation of the CUSUM Filter and AR Model-----	57
2.4.5	Event Correlation Engine-----	59
2.4.6	Java Optimization-----	61
2.4.7	Traffic Generation Tools-----	64
2.4.7.1	NetPerf-----	64
2.4.7.2	PSC TReno Server-----	64
2.4.7.3	ttcp and nttcp-----	64
2.4.7.4	NetPipe-----	65
2.4.7.5	Chariot-----	65

2.4.8	Software Test Sites-----	65
3.0	APPENDIX -----	66
3.1	RTNM Installation Instructions-----	66
3.1.1	Introduction -----	66
3.1.2	Software Required -----	66
3.1.3	Software Installation and Configuration-----	66
3.1.3.1	MySQL Installation and Configuration -----	66
3.1.3.2	JDK 1.3 Installation and Configuration -----	67
3.1.3.3	RTNM Installation and Configuration-----	67
3.2	Database Management Systems and Tools Evaluation Matrix-----	68
3.3	Math Package Evaluation Matrix -----	69
4.0	GLOSSARY -----	70

List of Exhibits

Exhibit 1.	Router Interface Series-----	3
Exhibit 2.	Arriving Octets at a Router Interface -----	27
Exhibit 3.	One-sided First-order CUSUM Pulses for Exhibit 2 -----	27
Exhibit 4.	One-sided Second-order CUSUM Pulses for Exhibit 2-----	28
Exhibit 5.	Combined Clusters from First and Second-order CUSUM Filters -----	29
Exhibit 6.	UML Class Diagram for CUSUM Filter -----	30
Exhibit 7.	UML Interaction Diagram between CUSUM and Its Clients -----	31
Exhibit 8.	UML Class Diagram for AR Modeling-----	32
Exhibit 9.	UML Interaction Diagram between AR and Its Clients-----	33
Exhibit 10.	RTNM Data Collection Overview -----	34
Exhibit 11.	The Network View -----	35
Exhibit 12.	The Node Monitor -----	36
Exhibit 13.	Database Browser-----	36
Exhibit 14.	Screenshot of Ping and Traceroute Tools -----	37
Exhibit 15.	The SNMP Trap Browser-----	39
Exhibit 16.	Bayesian Belief Network -----	41
Exhibit 17.	Operating Space as a Topological Space -----	44
Exhibit 18.	Topological Operating Space with Time Dimension -----	45
Exhibit 19.	Jade Platforms for Network Management-----	46
Exhibit 20.	UML Collaboration Diagram for AR Modeler -----	54
Exhibit 21.	Class Diagram for Topological Space O_E and Model Evolver-----	56
Exhibit 22.	Class Diagram for Monitoring Agents-----	57
Exhibit 23.	Event Correlation Architecture -----	61
Exhibit 24.	RTNM Portal -----	62
Exhibit 25.	RTNM Network View -----	62
Exhibit 26.	RTNM Node IP View -----	63
Exhibit 27.	RTNM Tools -----	63

List of Tables

Table 1.	AR Models for Epochs E_1 and E_2 of Exhibit 1 -----	4
Table 2.	AR Models for Epochs E_1^{in} and E_2^{in} of Exhibit 1 -----	4
Table 3.	Spectral Decomposition of Models E_1 and E_2 -----	5
Table 4.	Spectral Decomposition of Models E_2^{in} -----	5
Table 5.	Parameters for E_1^{in} Using the First 20 Data Points -----	17
Table 6.	Parameters for E_1^{in} Using the First 30 Data Points -----	18
Table 7.	Parameters for E_1^{in} Using the First 40 Data Points -----	18
Table 8.	Evaluation of Management Java APIs -----	38
Table 9.	Comparison of Software Design Tools -----	51

1.0 INTRODUCTION

This document is the Final Scientific and Technical Report (Contract Data Requirements List [CDRL] A004) for contract No. DAAH01-99-C-R129 entitled, "Real-Time Network Management." This is the final technical report for a 31-month effort (30 months research and 1 month for the final technical report), which ran from 14 April 1999 to 12 November 2001. BAE SYSTEMS Portal Solutions Inc. (PSI), formerly Synectics Corporation, was the prime contractor with the State University of New York Institute of Technology (SUNY IT) as a subcontractor on the effort.

In response to an invitation from Hillarie Ormon, the Phase I Defense Advanced Research Projects Agency (DARPA) sponsor of the Real-Time Network Management Small Business Innovative Research (SBIR) effort, Synectics Corporation submitted a proposal for Phase II. The Phase II SBIR contract was awarded on 14 April 1999 with the objective of developing methods for diagnosing network performance problems in real time and expanding upon the work accomplished in Phase I.

2.0 OBJECTIVES

The objective of this effort was to develop methods for diagnosing network performance problems in real time. According to our Phase I research, it is possible to collect data on the network and morph it into queuing models to produce information about the network and physical layers of nodes on a network. The Phase II research expanded upon the work accomplished in Phase I.

The program consisted of four tasks. The first task entailed the research and categorization of models that provided both a sufficiently accurate description of the behavior of the computer network and were computationally tractable for incorporation into a real-time network monitoring system. Secondly, PSI researched mechanisms for achieving the real-time determination of the parameters of the current-to-be models of a network, for monitoring k-step ahead forecasting accuracy of the current models, and for replacing the latter in case of breakdown. Thirdly, PSI built an object model of the analytical and operational entities that make up the architecture. Finally, PSI developed the application based on the research and design of the previous tasks using Java, with support from JMX (Java Management Extensions), JDBC (Java Data Base Connectivity), and the object model designed in the second task.

2.1 TASK 1 – MATHEMATICAL MODELING

2.1.1 NETWORK TRAFFIC MODEL RESEARCH

The range of computer network traffic models spans a rather wide spectrum, from the simple Markovian families at one end, to the newer self-similar long-range dependent varieties at the other end. It now appears that the accuracy of a model is fundamentally tied to the time scale on which the model is exercised. It has been demonstrated in the research literature that Markovian models are not accurate enough for large time scales, and that they therefore underestimate the resources required by the network over such time scales. However, the lower bound of a large time scale is not exactly clear; it is suspected to be in the minutes. On the other hand, models with long-range dependence should definitely not be applied to shorter time scales. Their computational requirements will exclude them anyway for real-time monitoring purposes.

A computationally efficient and theoretically tractable *compromise* between the two extremes is the *locally stationary* models. By this we mean concatenations of stationary but not necessarily Markovian models. In this approach, a fundamental problem centered on the identification of model change points. Indeed, it was expected (and confirmed in much of the computation we have carried out so far) that traffic may be subdivided into short epochs of just a few minutes duration, that may be assumed stationary. The problem of identifying the beginning of a new epoch is addressed in Sections 2.1.2 and 2.1.3. We modeled each epoch as a multivariate (vector) AR model, and recomputed the parameters of such a model for each epoch of the traffic. A stationary traffic AR model, where the traffic is specified as an m -dimensional vector time series $\{X_t\}$ (the m components correspond, for example, to m SNMP variables of interest), is given by the equation

$$X_t = W + \sum_{i=1}^p A_i X_{t-i} + \varepsilon_t \quad (1)$$

where p is the model order, the m -by- m matrices A_i are the p coefficients matrices of the model, W is the m -by-1 intercept vector, and ε_t is an m -by-1 random noise vector. We assumed that ε_t has zero mean with a covariance m -by- m matrix C . X_t itself is, of course, an m -by-1 vector. The set

$$\{p, A_i, W, C\} \quad (2)$$

will be referred to as the parameters of the model.

The modeling steps of both off-line analysis and real-time monitoring consisted of the following tasks.

- ☐ Identifying the next model change point
- ☐ For the next epoch
 - ◆ selecting an optimal model order p (in this work, we used a kind of Schwarz Bayesian Criterion statistic)

- ◆ estimating the parameters $\{ p, A_i, W, C \}$ of the model, and their confidence intervals (we used mostly least-squares techniques)
- ◆ performing a diagnostic checking (validation against observed data) of the fitted model (we used statistics like the Li-McLeod portmanteau criterion)
- ◆ performing spectral analysis of the fitted model

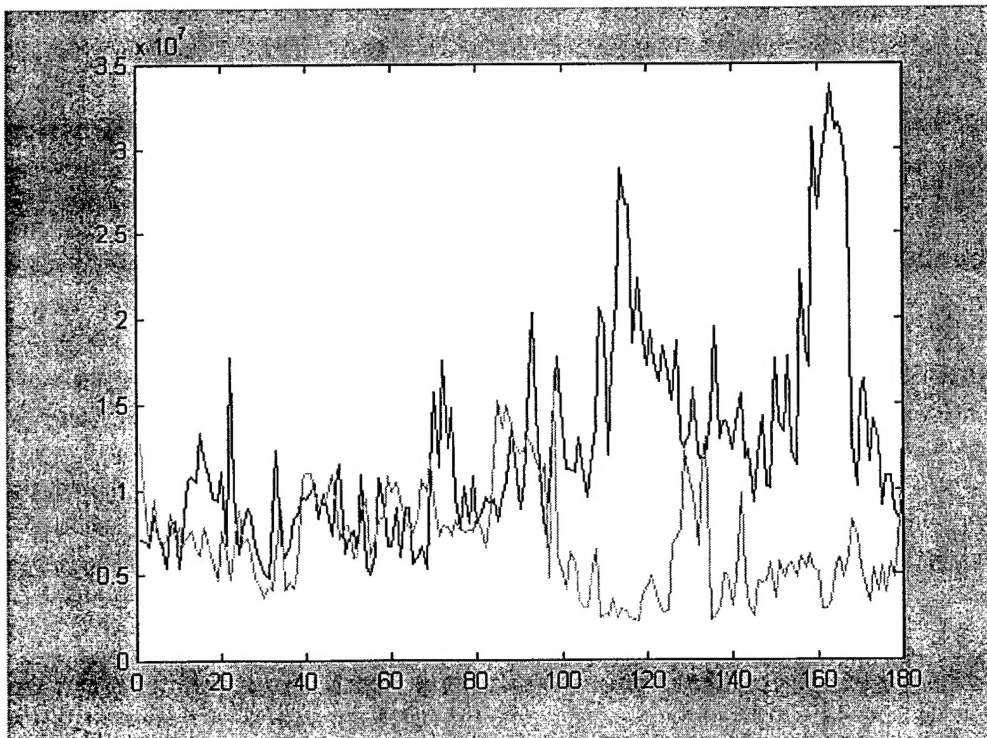
The end goal of the modeling process was a characterization of the underlying physical processes that give rise to the observed and modeled traffic. Such characterization could then be applied to more practical concerns, for example to:

- defining a concept of normal behavior, and thus characterizing (within some confidence intervals) that an observed behavior is anomalous
- assessing queuing parameters for further analyses and network resource requirements
- predicting the next vector value X_{t+1} using equation (1) above, as long as no new epoch has been signaled. For example, the expected value for X_t is given by the following equation, where I is the m-by-m identity matrix:

$$E(X_t) = (I - A_1 - A_2 - \dots - A_p)^{-1}W \quad (3)$$

Consider, for example, the following two-dimensional data collected at a router interface over a period of 15 minutes. In the graph black represents the ifInOctets and gray the ifOutOctets.

Exhibit 1. Router Interface Series



The Cumulative Sum (CUSUM) model change point technique of Sections 2.1.2 and 2.1.3 identifies $t \cong 115$ and $t \cong 165$ as change points. Here, we argued that what was observed at time t is the state of the interface, and no shift of the ifOutOctets data was attempted to match against ifInOctets. The modeling of epochs

$$E_1 = \{t \mid 1 \leq t \leq 115\} \text{ and } E_2 = \{t \mid 116 \leq t \leq 165\}$$

produces the following parameters.

Table 1. AR Models for Epochs E_1 and E_2 of Exhibit 1

	p optimal model order	A_1 coefficient matrix	W intercept vector	C noise covariance matrix
E_1	1	$\begin{bmatrix} 0.7310 & -0.0524 \\ -0.0701 & 0.7142 \end{bmatrix}$	$10^6 *$ $\begin{bmatrix} 3.3658 \\ 2.8888 \end{bmatrix}$	$10^{12} *$ $\begin{bmatrix} 9.8485 & -0.4018 \\ -0.4018 & 4.5499 \end{bmatrix}$
E_2	1	$\begin{bmatrix} 0.8392 & -0.0988 \\ -0.0357 & 0.4788 \end{bmatrix}$	$10^6 *$ $\begin{bmatrix} 3.5977 \\ 3.6242 \end{bmatrix}$	$10^{13} *$ $\begin{bmatrix} 1.8529 & -0.0793 \\ -0.0793 & 0.5571 \end{bmatrix}$

Note that modeling an m -dimensional vector time series is not equivalent to modeling m one-dimensional time series separately. For example, modeling only the ifInOctets series of Exhibit 1, we found for the two epochs

$$E_1^{in} = \{t \mid 1 \leq t \leq 115\} \text{ and } E_2^{in} = \{t \mid 116 \leq t \leq 165\}$$

the following parameters.

Table 2. AR Models for Epochs E_1^{in} and E_2^{in} of Exhibit 1

	p optimal model order	A_i coefficient matrices	W intercept vector	C noise covariance matrix
E_1^{in}	1	$A_1 = [0.7430]$	$10^6 *$ $[2.7861]$	$10^{12} *$ $[9.5992]$
E_2^{in}	3	$A_1 = [0.4356]$ $A_2 = [0.1121]$ $A_3 = [0.5600]$	$-10^6 *$ $[1.1331]$	$10^{13} *$ $[1.1677]$

For E_1^{in} the Li-McLeod portmanteau statistic evaluates to 0.0501, hence the fitted model passed the diagnostic checking defined by this statistic (the wisdom value for passing is 0.05). Likewise, E_2^{in} passed with a value of 0.05895.

The A_i and C parameters may be used to compute the spectral decomposition of the AR model. In the case of order $p = 1$, the computation was straightforward. For example, models E_1 and E_2 above have the following spectral decomposition, with corresponding confidence intervals.

Table 3. Spectral Decomposition of Models E_1 and E_2

	model spectral basis		confidence intervals for spectral basis	
E_1	$\begin{bmatrix} 0.7045 \\ -0.7097 \end{bmatrix}$	$\begin{bmatrix} 0.6013 \\ 0.7991 \end{bmatrix}$	$\begin{bmatrix} 0.4093 \\ 0.4063 \end{bmatrix}$	$\begin{bmatrix} 0.5995 \\ 0.4511 \end{bmatrix}$
E_2	$\begin{bmatrix} 0.9954 \\ -0.0960 \end{bmatrix}$	$\begin{bmatrix} 0.2581 \\ 0.9661 \end{bmatrix}$	$\begin{bmatrix} 0.0144 \\ 0.1498 \end{bmatrix}$	$\begin{bmatrix} 0.6533 \\ 0.1745 \end{bmatrix}$

In the case of order $p > 1$, the model was reinterpreted as an artificial model of order 1, with augmented block p -by- p coefficient matrix A_1' whose first block row consisted of the coefficient matrices of the original model. The $p-1$ subsequent block rows were simply the p -by- p block identity matrix (each diagonal block is the ordinary m -by- m identity matrix). Likewise, the state and noise vectors used with this A_1' were simply mp -by-1 vectors X_t' and ε_t' , where the rows of X_t' were $X_t, X_{t-1}, \dots, X_{t-p+1}$ in that order, and those of ε_t' were $\varepsilon_t, 0, 0, \dots, 0$ in that order (each 0 is the m -by-1 zero vector). From this reinterpretation, we could, for example, compute a spectral decomposition for the order-3 model E_2^{in} above.

Table 4. Spectral Decomposition of Models E_2^{in}

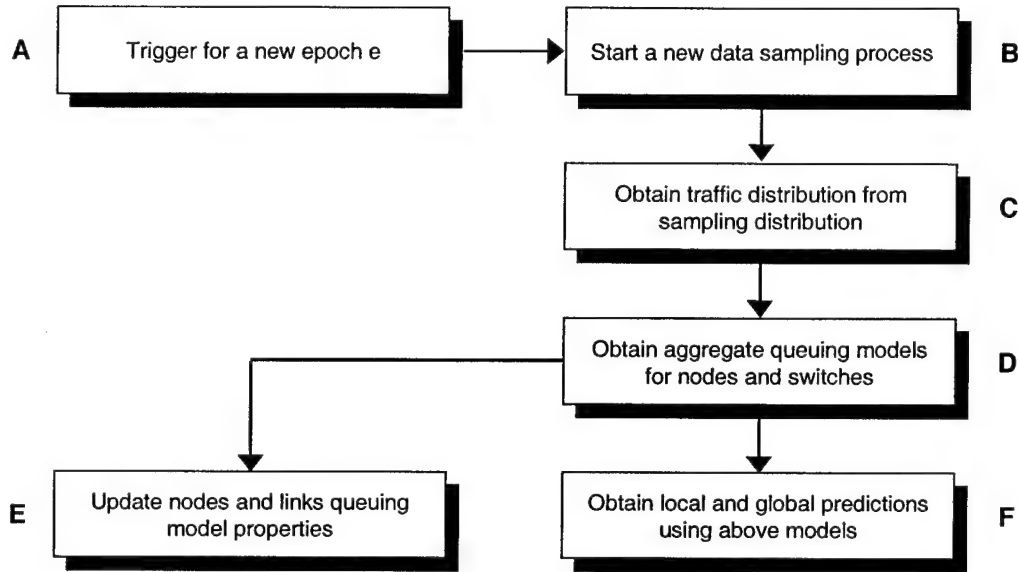
spectral basis	$[0.5489]$	$[-0.7378 - 0.0760 i]$	$[-0.7378 + 0.0760 i]$
Corresponding confidence intervals	0.0257	0.0563	0.0563

The real challenge lay in assigning dynamical semantics to the spectral information. For example, it might be feasible to classify the eigenvalues according to their network traffic importance.

2.1.2 MODEL CHANGE POINT RESEARCH

The detection of model-change points was paramount to our appropriate visualization of the network as a coherent system. Suppose at time t , a model point change is observed (for example, $M_\alpha(R|t) \wedge M_\beta(R|t+\delta)$ is true). Even though we might not know the model $M_\beta(R|t+dt \in e)$, we

could assume that until the next epoch e was indicated every sampling from then on should cohere for the new model that had to be computed from the sampling data. In other words,



In this section we outline our process for the box marked C above. The algorithm is described below. On an epoch e , we computed the relevant distribution parameters for the random variable X (based on the random samples collected within the epoch) and inferred the attendant distribution as shown.

- Obtain the sample mean $\bar{x} = \sum x_n / n$ on a sample density of n points.
- Obtain the sample variance $s_x^2 = \sum_{i=1}^n (x_i - \bar{x})^2 / (n-1)$
- Obtain the sample standard deviation $s = \sqrt{s_x^2}$
- Switch ($C_x^2 = \frac{s_x^2}{\bar{x}^2}$)
 - ◆ Case (1): **exponential** distribution with a mean $\bar{x} = 1/\lambda$, and a variance $s_x^2 = 1/\lambda^2$.
 - ◆ Case ($1/k$): **Erlang k**-distribution, $k \geq 1$ an integer.
 - ◆ Case (>1): a **k-stage hyperexponential** distribution (see case B for construction of this).
 - ◆ Default: a **gamma**-distribution with parameters α and λ .

The distributions were indicated next. For the random variable X the pdf was given by the following cases of interest.

Exponential: $f(x) = \lambda \exp(-\lambda x)$, $x > 0$, $\lambda > 0$

$$= 0, \text{ otherwise} \\ \bar{x} = 1/\lambda, \quad s^2 = 1/\lambda^2$$

Erlang-k :
$$f(x) = \frac{\lambda k (\lambda x)^{k-1} e^{-\lambda x}}{(k-1)!}, \quad x > 0, \lambda > 0, k \text{ an integer}$$

$$= 0, \text{ otherwise}$$

$$\bar{x} = 1/\lambda, \quad s^2 = 1/(k\lambda^2)$$

Suppose, for our distribution $\lambda = \bar{x}$. Let k be the largest integer less than or equal to (\bar{x}^2 / s_x^2) , i.e., the floor of this. Then this distribution is the Erlang- k random variable with parameters k and λ .

k-stage hyperexponential:
$$f(x) = \sum_{i=1}^k \alpha_i \mu_i e^{-\mu_i x}, \quad x > 0, \mu_i > 0, k \text{ an integer}$$

$$= 0, \text{ otherwise}$$

we compute $E[x] = \bar{x} = \sum_{i=1}^k \frac{\alpha_i}{\mu_i}, \quad E[x^2] = 2 \sum_{i=1}^k \frac{\alpha_i}{\mu_i^2} \quad s^2 = E[x^2] - E[x]^2$

This arose when packet arrivals, for instance, tended to form clusters or service function was approximately k -parallel exponential stages of active servers. A typical switch could depict such a distribution.

gamma:
$$f(x) = \frac{\lambda (\lambda x)^{\alpha-1} e^{-\lambda x}}{\Gamma(\alpha)}, \quad x > 0, \alpha > 0, \lambda > 0$$

$$= 0, \text{ otherwise} \quad \Gamma(x) = \int_0^\infty x^{t-1} e^{-x} dx, \quad t > 0$$

$$\bar{x} = \alpha / \lambda, \quad s^2 = \alpha / \lambda^2$$

The results of the AR modeling of Section 2.1.1 were effectively used to compute the statistics (for example, mean and variance) of the above algorithm. For example, equation (3) provides an estimate of the means from the model.

2.1.3 QUEUING MODEL RESEARCH

For queue modeling, we looked at applicable M/G/1 and G/M/1 formulations, and more generally at (possibly Markov-modulated) G/G/1 regimes. The models of Section 2.1.1 approximately provided G, the general arrival or service distribution. The main issue was, of course, the (real-time) computation of the statistics of the queue.

For M/G/1, the so-called Pollaczek-Khinchin mean-value formula provided the average number of users in the system, viz.

$$r = \rho + \rho^2 \frac{1+C^2}{2(1-\rho)}$$

with C being the coefficient of variation of the service time (probability distribution), and ρ the load (average arrival rate divided by the average service rate). In particular, if $G = D$ (deterministic service time), then $C = 0$, so that

$$r = \frac{\rho(2-\rho)}{2(1-\rho)}$$

Taking a deterministic service time might be, in many cases, a good approximation, as it was quite difficult to evaluate the exact service time distribution. The variance ν and higher-order statistics of the number of users in the system could be obtained by so-called transformation methods. They involved, however, higher-order moments of the service time distribution. We were looking at numerical evaluations of these higher-order statistics.

A G/M/1 is the dual of M/G/1, since the statistics of the arrival time distribution account for the waiting time statistics. The duality took us again to Laplace transforms. In particular, the parameters r and ν involved the Laplace transform of the arrival time probability distribution. Indeed, if θ was the solution to the equation

$$\theta = \Lambda(\mu - \mu\theta)$$

(θ is the probability of an arrival finding the queue full), where $\Lambda(z)$ was the Laplace transform of the arrival distribution G , and μ was the assumed Markovian service rate, then

$$r = \frac{\rho}{1-\theta}$$

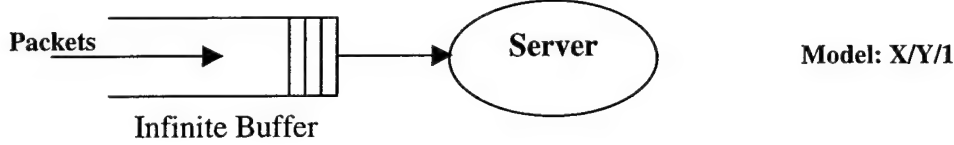
with variance

$$\nu = \frac{\rho(1-\rho+\theta)}{(1-\theta)^2}$$

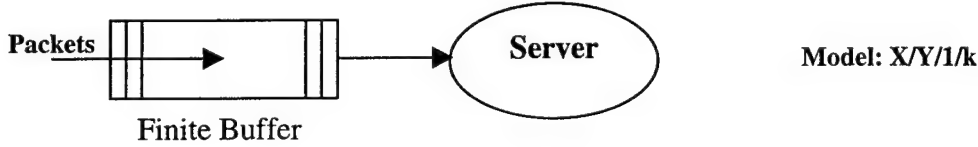
The main issue with the implementation of these formulas lay in the numerical estimation of θ , ρ and μ . We showed next how to compute the service rate and effective buffer size using an iterative algorithm.

One of the problems that required immediate attention was ascertaining a server's service rate from posted data on traffic. Without this, a queuing analysis would be impossible; any interesting property vector of a server had to include this as a parameter. Also of importance was the knowledge of the available buffer volume at the server. If such a datum were available, it would be easier to control and plan its action. Every server has to have some storage for incoming packet traffic desiring service. If the server is highly utilized, it is possible that its storage capability is reduced to naught in which case it would begin to drop incoming packet traffic as it came to the server for service.

Suppose a server node was configured with b MB of buffer storage. Even though it was known that the queuing volume of a server could not exceed b MB it was not easy to determine the effective capacity of the buffer due to variations in packet size. Consider a queuing system $X/Y/1/k$ depicting a network node (a router, a host) offering a total of k packet buffers for temporary storage of packets. As long as packets were not dropped it would appear as



However, if the same node began to drop packets owing to buffer unavailability (full buffer), it would appear, given everything else as,



Therefore, from a server's performance, particularly as it began to drop packets from entering into buffer, we would have to infer the server's effective storage capacity. Let us suppose the storage capacity was k packets, k being the maximum number of packets the buffer can hold at one time. For convenience, we assumed a $M/M/1/k$ system. The probability that the server's buffer was full, i.e., the packet dropping probability was

$$p_k = p_{drop} = \frac{(1-\rho)\rho^k}{1-\rho^k} \quad (4)$$

where the traffic intensity at the buffer before it became full was $\rho = \lambda / \mu$. Let us assume that the packet arrival rate $\lambda(t)$ was known but $\mu(t)$, the rate at which the server-transmitted packets onto the network was not known but must be inferred (the general case). Suppose at two consecutive sampling instances, the server rates were, on average, $\mu(t)$ and $\mu + \delta\mu(t')$. Accordingly, Equation (4) could be expressed as

$$(1-p_k)\rho_t^{k+1} = \rho_t^k - p_k \quad (5a)$$

$$(1-p_k)\rho_{t+\delta}^{k+1} = \rho_{t+\delta}^k - p_k \quad (5b)$$

Therefore,

$$\frac{\rho_{t+\delta}^{k+1}}{\rho_t^{k+1}} = \frac{\rho_{t+\delta}^k - p_k}{\rho_t^k - p_k} \quad (6a)$$

In this, we were making the assumption that the probability that a packet coming for service would find the buffer full remains, on the first order approximation, more or less the same. Equation (6a) could be also be expressed as

$$\frac{(\rho + \delta\rho)^{k+1}}{\rho^{k+1}} = \frac{(\rho + \delta\rho)^k - p_k}{\rho^k - p_k}$$

This could be further simplified as

$$\left(1 + \frac{\delta\rho}{\rho}\right)^{k+1} = 1 + \frac{k\rho^{k-1}\delta\rho}{\rho^k - p_k}$$

which in turn, in the first order approximation, leads to

$$\rho^k = (k+1)p_k \quad (6b)$$

In terms of this, Equation (4) can be re-expressed as

$$1 - \rho^{k+1} = (1 - \rho)(k+1)$$

from which we get a recursive computation

$$\rho_{next} = 1 - \frac{1 - \rho_{last}^{k+1}}{k+1} \quad (6c)$$

whence, the derived service rate $\mu(t)$ is obtained, when (6c) converges,

$$\mu(t) = \bar{\lambda} / \rho \quad (7)$$

where $\bar{\lambda}$ is an epoch average arrival rate.

This we defined as the Equivalent Markovian Server (EMS) rate of our server where the effective service distribution may not be exponential. But, in order to use (7) we needed to know the maximum number of buffers with which a server was endowed, i.e., the value of k that does not explicitly involve the variable μ . Note that in this model, the functional size k always obeyed the recursive expression

$$\frac{1}{k} \log\{p_k(1+k)\} = \log\left(1 - \frac{1}{1+k}\right) + \log(1 - p_k) \approx \left(\frac{1}{1+k} + \frac{1}{2(1+k)^2} + \dots\right) + \log(1 - p_k)$$

from which we obtained

$$k = \frac{\log(1+k) + \log p_k}{\sum(k) + \log(1 - p_k)}, \text{ and } \sum(k) = \frac{1}{1+k} + \frac{1}{2(1+k)^2} + \frac{1}{3(1+k)^3} + \dots \quad (8)$$

Using an average of k as obtained in (8) over the sampling distribution we could arrive at an effective estimate of k . The algorithm to be followed could be cast as

- ☐ Start with a good estimate (initial value) of k . A typical approach would be $k_{init} = f(\text{queue_length} | p_k \neq 0)$
- ☐ Use (8) to get an estimate of k when it converges

- Use the average of all estimates over some sample runs

The significance of Equations (7) and (8) could be appreciated as follows. If the true performance statistics of a server were not known *a priori* (for instance, SNMP does not advance any clue), then a maximum entropy distribution conjectured at a level of maximum ignorance would always lead to an exponential distribution for the server performance. In this sense, an effective EMS measure projecting a server's service rate was always a good choice.

In deriving appropriate queuing models for network entities, such as routers, switches, etc., one had to obtain a reasonable estimate of the server's service rate. For every Simple Network Management Protocol (SNMP) data poll, the current utilization calculation needed to be based on some maximum throughput in order to arrive at a percentage. This allowed us to gauge how heavily any given node was utilized.

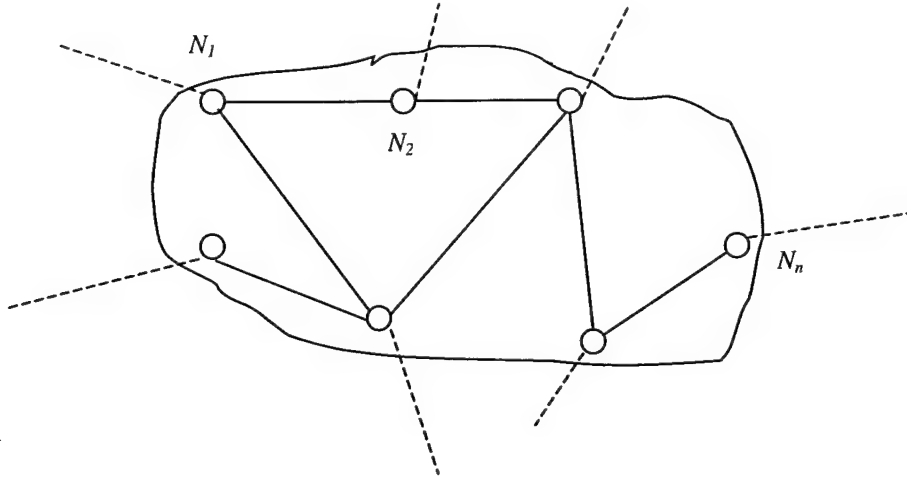
Since SNMP-collected data on these parameters did not exist, we had to take a more relativistic approach. For each node, a maximum throughput was learned through long-term polling and service rate calculations. If we discovered a service rate that was higher than what we thought, the current value simply replaced the old. After a reasonable amount of time (e.g., 24 hours), we could state that our maximum service rates were reasonably accurate. Therefore, each poll's service rate was compared to the maximum service rate, and we arrived at our percentage. Each service rate was relative to that node's past performance.

Although this approach did not yield a definitive maximum service rate for a node, it did provide the network manager with perhaps a more useful statistic: the traffic level on any given node relative to its standard traffic level.

2.1.4 AGGREGATE MODEL RESEARCH

A natural aggregate model for any subnetwork (or network) of n nodes N_1, N_2, \dots, N_n (e.g., n routers), extending the node model of Section 2.1.1, is a multivariate AR model, of dimension $m = nq$, where q is a number of variables of interest at each node (e.g. q SNMP variables). The simplest example of such a model corresponded to the case $q = 1$. For example, we might be interested in the overall packet load distribution in a network of n routers, and choose, as single subdimension at each node (router), the sum of two SNMP variables.

$$\text{ifPkts} = \text{ifInUcastPkts} + \text{ifInNUcastPkts}$$



Note that this modeling approach applies to local area networks as well. In any case, the machinery of Section 2.1.1 carries over to networks and subnetworks. In theory, it is not even required that the subdimension be the same for all nodes. We might, for example, be interested only in a few variables in the “interior” of our network, but require more information (thus higher subdimension) at the “periphery”. Hence, in general, the total dimension of the multivariate AR model is

$$m = \sum_{i=1}^{i=n} q_i$$

where q_i is the subdimension at node N_i .

The proposed model scaled naturally. New nodes simply contributed additional dimensions to the model. The main issue remained the identification of model change points. Our approach was to perform CUSUM filtering, as explained in Section 2.2, at each node of the aggregate model, then take the union of the so identified change points, eliminating points of low significance. We noted that physical characteristics, such as clock discrepancy and wire distance were, in some sense, already built into the vector AR model. Of course, the model computation complexity scaled up with the size of the network. We collected and analyzed (as in Section 2.1.1) traffic data from multiple routers and switches.

We used Traceroute and Ping to build a network topology model. The network itself was seen as connectivities among nodes and switches along with some kind of performance measure either and/or on the connecting links and individual participating nodes, respectively. This provided a logical architecture of the network. Therefore, if we had n nodes in a communicating network, the overall systemic topology $\mathfrak{S}(net)$ was an aggregate collection of individual topologies rationalized at a meta level where the entire network topology $\mathfrak{S}(net)$ was visualized. Therefore, the network topology was defined as

$$\mathfrak{S}(net) = \bigcup_c \mathfrak{S}(i | i \in nodes) \quad (9)$$

where \cup_c was a union operation of all the individual network topologies $\mathcal{S}(i|i \in \text{nodes})$ seen from distinct active nodes $i \in \text{nodes}$ with the subscript c referring to list update operation such that the overall view $\mathcal{S}(i|i \in \text{nodes})$ was internally consistent within the topology list.

Internally, from a specific node k , the topology was the set $\mathcal{S}(k)$ comprising minimally the following:

- From the node k the direct links to all other neighboring nodes was seen along with the transmission delay and its associated variance, respectively, as obtained in Algorithm 1 and Algorithm 2 below using Ping/Traceroute routines.
- The congestion levels of the neighboring nodes of k through Algorithms 3 and 4 depicted the node-level congestion, regional-congestion (within 1-hop distance), and district-level congestion (within h -hop distance) in either an optimistic or a pessimistic frame of reference.

The set $\mathcal{S}(k)$ was finally realized as a vector in Algorithm 5. For our purpose, we allowed the user (also known as a controller/manager) to define the dimension of the district either by selecting a region around a node, or setting the maximum hop length to a node within the district. Also, in all algorithms below, the one central measure was the transmission time or a sojourn time of a probing packet from a node k to a node l introduced as a property of the link (kl) connecting the two nodes viz. $\{ \text{link}(kl), r(kl), l \in \text{nodes} \}$, where $r(kl)$ was a property vector with the $\text{link}(kl)$. For the time being, we assumed it to be a scalar indicating an ICMP probe-packet transmission time from node k to node l , $t(kl)$. Since over all of these sampling events this measure could yield a minimum of three estimates of $r(kl)$, namely best time, worst time, and the average time measure for any pair of nodes k and l , we could have three topology estimates of the network from any specific node. Let these topologies from a node k be $\mathcal{S}_b(k), \mathcal{S}_w(k)$ and $\mathcal{S}_a(k)$, respectively.

2.1.4.1 ALGORITHM 1

From manager node k , the agent identified the list of all neighbors of k one hop-away. Let the list comprise the addresses $\langle IP_1^k, IP_2^k, \dots, IP_i^k, \dots, IP_m^k \rangle$. This list was updated every T seconds. Each node IP_i^k in the above list sent its own neighborhood list $\langle IP_1^i, IP_2^i, \dots, IP_j^i \rangle$ to the manager node k every T seconds.

The manager node k , using all such lists, obtained the global neighborhood lists as a union over all these individual neighborhood lists. This list was always self-consistent. The global list was maintained as a collection of adjacency lists implemented as arrays.

For each item IP_α^k in the global list within a distance of h hops from the manager node k , its neighborhood was obtained from the next round collection and appended to the existing global list. After kT seconds, step 1 was repeated to update its list.

This algorithm would be used to produce the best time, worst time, and the average time topology for the network evolving from the manager node in the network. These time estimates could be obtained either with Traceroute or Ping.

2.1.4.2 ALGORITHM 2

This algorithm would be basically the same as Algorithm 1 except that now we would measure variance σ_{kl}^2 of the random variate $t(kl)$ provided node l was within h hops of the manager node k . However, in this case we would have only a single variance topology map. Either Ping or Traceroute could be used to obtain delay samples between the probing node and the probed node. To avoid the outliers, we discarded 10% of the best estimates, and 10% of the worst delay estimates to compute the variance on the remaining samples.

2.1.4.3 ALGORITHM 3

Using Ping and/or Traceroute, the number of packets dropped at each node with a packet size set at p was obtained. The default packet size was 64 bytes. However, since we were interested in realistic cases, the packet size could be set at the average packet size the IP layer usually handles at the probing node. Using this option a packet-drop map at the nodes of topology list was obtained. Let the computed measure of the count of number of packets dropped at a node l up to time t since the last count at an earlier time $drop_l(t)$ at a node l obtained at a time t . In terms of the number of packets actually served $serv_l(t)$ at the node l , the probability of a packet being dropped at an arriving node l could be obtained as

$$prob_{drop}(l|t) = \frac{drop_l(t)}{serv_l(t) + drop_l(t)} \quad (10)$$

This was a first level estimate of dropping probability at a specific node l . If the system were in equilibrium within an epoch e , all such probability estimates at a specific node would be sampling distribution of the population mean within the epoch. Accordingly, this algorithm yielded for a node l the population mean

$$p(l_{drop}) = \int_{t=0}^{t \in e} prob_{drop}(l|t) dt \quad (11)$$

This algorithm using the measure (11) for an estimate of drop probability could be used to obtain the regional congestion level at and around a specific node l . The congestion map could be defined in many ways, and there are many possible varieties of definitions. One likely definition that was suggested at this stage is seen in the next algorithm.

2.1.4.4 ALGORITHM 4

On a topology $G = (V, E)$ the congestion-map was conjectured as follows. At a node l , the node-congestion is

$$cong(l|node) = p(l_{drop}) \quad (12)$$

We defined the link-congestion as the ratio of actual amount of throughput (in bytes) realized on a link (i.e., the number of bytes moving through the link per unit time) against its available bandwidth B in bytes/sec. This information could be obtained from SNMP procured data. Let us define, accordingly,

$$cong(link(kl)) = X_{kl}(t) / B \quad (13)$$

where $X_{kl}(t)$ is the throughput on link kl at time t .

The regional congestion at a node l is then defined as follows:

Regional Congestion (optimistic)

$$cong(k | regional) = 1 - (1 - cong(k | node)) \times (1 - \min_l cong(link(kl))) \quad (14)$$

Regional Congestion (pessimistic)

$$cong(k | regional) = 1 - (1 - cong(k | node)) \times (1 - \max_l cong(link(kl))) \quad (15)$$

Of course such measures would be used with some appropriate correcting factors to align them properly with the given initial and boundary conditions.

The regional congestion map could be further aggregated to produce district congestion map at and around a given node. Either, the controller/manager selects the region around the node in question by marking it appropriately, or, by default, the district-region of a node k would be identified within two hops away from k . Therefore,

- ☐ **default-setting:** node l is within district of k , if $hop_dist(kl) \leq 2$
- ☐ **selected specification:** node(l) is in district(node(k)) if $node(l) \in marked_set(node(k))$
- ☐ **user specified district diameter:** In the setting box, the user specifies the diameter of the district. If the diameter is p , then node l is within district of k when $hop_dist(kl) \leq p$. Default setting is a special case of this with $p=2$.

The district congestion map would be either optimistic or pessimistic. The map would be constructed accordingly.

Mark the entire district around node(k) with a color or a number (between 1 to 10) directly proportional to the computed value as follows.

For an optimistic district congestion map around a node(k):

$$cong(k | district) = 1 - (1 - \min_l cong_l^{min} | regional) \times (1 - avg_l cong_l^{min} | regional) \quad (16)$$

For a pessimistic district congestion map around a node(k):

$$\text{cong}(k | \text{district}) = 1 - (1 - \max_l \text{cong}_l^{\text{max}} | \text{regional}) \times (1 - \text{avg}_l \text{cong}_l^{\text{max}} | \text{regional}) \quad (17)$$

2.1.4.5 ALGORITHM 5

This algorithm would collect the results obtained from all the above computations and obtain the property vector of each and every node l visible from the manager node k . Periodically this list would be updated, predicated by the sampling frequency of the traffic profiles around a node.

2.2 TASK 2 – REAL-TIME MODEL SELECTION

2.2.1 REAL TIME ESTIMATION OF MODEL PARAMETERS

The estimation of the model parameters of Section 2.1.1 is most accurate when all data for an epoch are available. In the real-time monitoring context, we may, of course, only use the data collected so far in the current epoch to estimate the epoch's model. We have conducted experiments to ascertain the effects of the data run length on the order p and the other parameters A_i , W , C of the models in order to come up with useable heuristics. We report in the following tables only the results for the E_l^{in} epoch of Section 2.1.1, but our findings for most other epochs show basically the same patterns, with a high degree of similarity. Table 5 shows the models computed by our algorithm when only the first 20 data points of the E_l^{in} epoch were used. The results differ in the maximum order inputted, which is a parameter of the algorithm.

Table 5. Parameters for E_1^{in} Using the First 20 Data Points

max order p proposed to algorithm	actual p computed	coefficient matrix A_i computed	intercept vector W computed	noise covariance C computed
3	1	[0.6836]	$10^6 * [3.0647]$	$10^{12} * [2.8155]$
4	1	[0.7100]	$10^6 * [2.7729]$	$10^{12} * [2.9568]$
5	3	[0.7507] [-0.3034] [0.3259]	$10^6 * [2.3690]$	$10^{12} * [3.1143]$
6	3	[0.7292] [-0.2746] [0.3015]	$10^6 * [2.3690]$	$10^{12} * [3.3997]$
7	5	$10^6 * [4.4684]$	$10^{12} * [3.1387]$
8	8	$10^6 * [-1.7930]$	$10^{12} * [4.9017]$

The maximum order p proposed to our algorithm cannot be too high compared to the data length r . In fact, when it approached $\frac{r}{2}$, then the model was over-fitted to the observed data, as can be seen from the high value of the actual computed order. The parameters A_i and W found were in this case consistent with the values shown in Table 2, which implies that the noise covariance would be inconsistent. Intuitively, the algorithm had to decide how much of the relatively meager data it saw was fitted or treated as noise. Tables 6 and 7, on the other hand, show that the data seen may be accounted as more of random fluctuations and not as trends.

Table 6. Parameters for E_1^{in} Using the First 30 Data Points

max order p proposed to algorithm	actual p computed	coefficient matrix A_i computed	intercept vector W computed	noise covariance C computed
3	1	[0.3097]	$10^6 * [6.0854]$	$10^{12} * [7.3223]$
4	1	[0.3137]	$10^6 * [6.0365]$	$10^{12} * [7.6227]$
5	1	[0.3096]	$10^6 * [6.1309]$	$10^{12} * [7.8598]$
6	1	[0.2956]	$10^6 * [6.3179]$	$10^{12} * [8.2502]$
7	0	none	$10^6 * [9.1548]$	$10^{12} * [3.1387]$

Table 7. Parameters for E_1^{in} Using the First 40 Data Points

max order p proposed to algorithm	actual p computed	coefficient matrix A_i computed	intercept vector W computed	noise covariance C computed
3	1	[0.3165]	$10^6 * [5.9284]$	$10^{12} * [6.3277]$
4	1	[0.3200]	$10^6 * [5.8861]$	$10^{12} * [6.5080]$
5	1	[0.3192]	$10^6 * [5.9316]$	$10^{12} * [6.6487]$
6	1	[0.3117]	$10^6 * [6.0341]$	$10^{12} * [6.8025]$
7	1	[0.2916]	$10^6 * [6.2931]$	$10^{12} * [6.7655]$
8	1	[0.2926]	$10^6 * [6.2826]$	$10^{12} * [6.9909]$
19	0	$10^6 * [8.3674]$	$10^{12} * [8.3501]$

Here, when the maximum order p proposed approached $\frac{r}{2}$, then the observed data were all treated as the noise ε_t , as can be seen from the zero order computed. The parameters A_i and W then had values somewhat inconsistent with those of Table 2. But, this was not as odd as it appeared, since the high value of the intercept vector accounted for the low value of the coefficient matrix. The heuristic derived from our experiments could be stated as follows.

- ❑ Within a new and same epoch, sample 20 times, if no change point had been signaled (see Sections 2.1.2 and 2.1.3), to compute the current model; input a low maximum order (3 or 4) to the algorithm; if so desired, recompute the model every δ samples (e.g. $\delta = 10$ or 20).
- ❑ If a change point was signaled, repeat with the approach stated above.

2.2.2 MODEL COMPARISON RESEARCH

A decision as to the adequacy of a model M_0 , based on the current value of some numerical quantity Q , usually referred to as the *monitor*, chooses between three alternatives.

- Δ_a acceptable model performance; reinitialize monitor as appropriate
- Δ_f further observation needed; update monitor accordingly
- Δ_s signal model performance inadequacy.

Initially, we wanted to concentrate on monitors utilizing so-called *Bayes' factors*, sometimes called *weights of evidence*, which may be defined as follows. Suppose we must compare two models M_0 and M_1 . At time t , suppose we accumulated some historical information D_{t-1} , if any, that may help us decide. At time t , suppose we have an observed quantity X_t (a random variable of interest, such as the SNMP ifInOctets variable). The Bayes' factor for M_0 versus M_1 , at time t , is the ratio

$$H_t = H_t(1) = \frac{p(X_t | M_0, D_{t-1})}{p(X_t | M_1, D_{t-1})}$$

That is the odds that X_t comes from M_0 against that it comes from M_1 , given D_{t-1} . More generally, if k consecutive observed values

$$X_{t-k+1}, X_{t-k+2}, \dots, X_{t-1}, X_t$$

are available, then the Bayes' factor for M_0 versus M_1 , at time t , is the product $H_t(k)$ of the quantities $H_m(1) = H_m$ above, for $m = t-k+1, \dots, t$. Clearly,

$$H_t(k) = H_t(1) H_{t-1}(k-1)$$

for $t > 1$, i.e., evidence for M_0 versus M_1 accumulates in multiplicative fashion, as new observations come in (or in additive fashion in log space). Let

$$h(k) = h(0) H_t(k)$$

where $h(0)$ is the initial condition, viz. the a priori odds ratio for M_0 versus M_1 . $h(k)$ may be used as monitor by prescribing thresholds α and β , with $\alpha > \beta > 0$, and by choosing

- Δ_s if $h(k) \geq \alpha$
- Δ_a if $h(k) \leq \beta$
- Δ_f otherwise

Experiments implemented in Matlab, and the real-time computation requirements for monitoring a large number of models (corresponding to nodes and links in a large computer network), have led us to reevaluate the use of Bayes' factors as monitor for computer network models. In

statistical process control, a common quantity used as monitor is the so-called CUSUM, or cumulative sum. We will expand on it in Section 2.2.3, but in this section, we show that its use in our network monitoring investigations may be justified both in practical and theoretical terms.

With Δ_a , Δ_f , and Δ_s defined above, let X_1, X_2, \dots, X_r be r observed values of the variable of interest, since a Δ_a or Δ_s decision was made. In other words, the r observed values correspond to a run of r Δ_f decisions. The CUSUM technique uses a pair of monitors referred to as upper and lower CUSUMs. Abstractly, an *upper* CUSUM decision scheme, denoted

$$\text{CUSUM}^+(A, \beta, \alpha)$$

uses the quantity

$$C_k = \sum_{i=1}^k (X_i - A) = C_{k-1} + X_k - A$$

as monitor, and chooses

$$\begin{aligned} \Delta_s & \text{ if } C_k \geq \alpha \\ \Delta_a & \text{ if } C_k \leq \beta \\ \Delta_f & \text{ otherwise} \end{aligned}$$

For the moment, think of A as a target value for a quantity being monitored, and of α and β , with $\alpha > \beta > 0$, as prescribed thresholds. A lower CUSUM scheme, denoted $\text{CUSUM}^-(A, \beta, \alpha)$, is defined in a similar fashion, with the monitor satisfying

$$C_k = C_{k-1} + A - X_k$$

On the surface, the above monitors appear unrelated. Consider, however, the family of exponential distributions, including normal, gamma, and Weibull. A member of this family has a natural scalar parameter η , and may be described by a probability distribution of the form

$$p(X, \eta) = \theta(X, \varphi) e^{\varphi(X\eta - a(\eta))}$$

for some convex function $a(\eta)$, and factor function $\varphi(u)$. Two exponential models that are identical except for the value of the natural parameter η may be compared by means of the Bayes' factor monitor defined above. Here, using r observations, we have

$$\begin{aligned} \log h(r) &= \log h(0) + \log H_r(r) \\ \log H_r(r) &= \varphi(\eta_0 - \eta_1) \sum_{i=1}^r \left[X_i - \frac{a(\eta_0) - a(\eta_1)}{\eta_0 - \eta_1} \right] \end{aligned}$$

The decision scheme, model defined by η_0 versus model defined by η_1 , described by this monitor, and using thresholds $\alpha > \beta > 0$, is then equivalent to $\text{CUSUM}^+(A, \beta', \alpha')$, where

$$A = \frac{a(\eta_0) - a(\eta_1)}{\eta_0 - \eta_1}$$

$$\beta' = \frac{\log \beta - \log h(0)}{\phi(\eta_0 - \eta_1)}$$

$$\alpha' = \frac{\log \alpha - \log h(0)}{\phi(\eta_0 - \eta_1)}$$

For example, comparing two normal distributions with unit variances and means μ_0 and μ_1 with a Bayes' factor monitor of thresholds $\alpha > \beta > 0$, is equivalent to comparing them using a CUSUM⁺(A, β' , α'), with $\beta' = 0$, $\alpha' = \frac{\log \alpha - \log h(0)}{\mu_0 - \mu_1}$ and $A = \frac{\mu_0 + \mu_1}{2}$.

For general distributions, we do not expect that CUSUM and Bayes' factor decision schemes are equivalent. However, given the simplicity and efficiency of CUSUMs, and given the theoretical equivalence for exponentials, the use of CUSUMs is justified.

2.2.3 MODEL CHANGE POINT IDENTIFICATION

We distinguished two interrelated cases of change point estimation for network data. The non-sequential case was the off-line problem described as follows. The network data $X(t)$ for a time interval $[0, T]$ was available and assessed as a multivariate model parameterized by a set S of parameter matrices and vectors, for $t \in [0, T]$. The problem was then to test the null hypothesis of no change in S versus the alternative hypothesis that there existed $x \in (0, T)$ such as there was a change in the underlying process generating the data in (x, T) . The sequential case was the on-line real-time problem where the network data arrived sequentially. The change point should be recognized as soon as possible after it occurred, since the false detection rate needed to be as low as possible. The above two versions of the change point analysis problem were much related in that the off-line analysis could provide insight into the "distribution" of change points in the network data.

We investigated various ways of improving the model of Section 2.1.1 for real-time computation and assessment. In particular, we looked at a possible use of principal component expansions of multivariate time series. Such expansions write the original time series as a sum of a small number of components $X(t) = A_1(t) + A_2(t) + \dots + A_n(t) + \varepsilon(t)$. The A_i were assumed to be independent in some sense, and also interpretable; $\varepsilon(t)$ as a random noise component. Interpretability here related to factors such as trend, cyclicity, and other harmonic interpretations. For the off-line problem, we looked at wavelet-based techniques as well, and compared with them with the spectral outputs of Arfit.

Given the data we collected so far, it might actually be more reasonable to implement a tamer problem, namely, that of the detection of abrupt changes in the mean only. For this problem, sophisticated tools like wavelets and principal component expansions may offer little improvement over traditional techniques like the Kolmogorov-Smirnov tests for change in distribution and for change in spectrum.

The model abstraction problem was explored from a real-time network environment perspective, usually admitting different system models (such as queue-theoretic) at its different equilibrium states. To computationally depict system states at any level of abstraction, it was necessary to identify correct models consistent with observables. However, any such system identification need not be permanent, particularly for a dynamic system. In such situations, as the system appears to migrate from one equilibrium state to another, one should be able to quickly identify an event of context switching from one model abstraction to another. In this section, we show how, using a variation of traditional CUSUM statistical approaches, one could identify model change events on time.

Our objective was to explore the problem of model abstraction for a dynamical system such as computer networks, to be realized in real-time so that it could be monitored and controlled effectively. If a system were capable of being in only one equilibrium state, we would have the simpler problem of deriving a consistent system model given its observed performance values such as those appearing as SNMP variables. However, a network is capable of being in a number of plausible equilibrium states. Even if we conceded that we would concern ourselves primarily with queuing models at a network node or a link (client-server type queuing models requiring buffer), we would have a number of different models from which to choose. The distribution types of the traffic rates would undoubtedly predicate the functional model. A M/M/1 queuing model at a server node could suddenly change to a G/M/1/k type model, and unless we captured this model-change event on time we would almost surely have a wrong profile of the attendant node. Ideally then it was essential to find out when and to what kind of model a given system migrates as it moves from one equilibrium state to another. This was attempted in this section

Our observables were all SNMP brokered time-series; using these we should be able to realize a specific parametric system model. For instance, at an interface where a node was connected to the communication net, we measured an observable like "IfOutOctets" depicting the amount of octets sent out at the interface by the server since the last observation. These observables were our only clue to infer profiles of individual nodes and links, to infer the entire network as a function of these SNMP-supplied RMON variables. The next logical step was to devise a way to ascertain a model change point.

For our purpose, we considered a real-time dynamical system R whose observed state s changed in a non-deterministic fashion over time. Our objective was to depict R by a predictive model $M(R)$ on a real-time basis through its observable s . If at a time t the underlying system model was $M(R)$ we should be able to predict, in view of this model, its system's state $s(t')$ at a nearby time $t' = t + \delta t$ given that we knew its system state s at an earlier time t .

However, a dynamical system may change its behavior during its existence. A change in system behavior may not always be explained in terms of the current system model and it might be necessary to consider a different model, which may help understand the current system behavior better. In view of this, we needed to realize a procedure to pinpoint the time instance when a system changed its model from $M(R)$ to $M'(R)$. When the underlying model affecting the system's behavior changed, the result was a system state change. However, the reverse was not necessarily true. Not all state changes would be an indication of a model change. Within the scope of $M(R)$ a change $s \rightarrow s'$ might be a valid observation and it might not point to any model change at the system level. Our objective in this section was to obtain a systematic approach where we would be able to decipher two types of events from each other, namely (we used the symbols \rightarrow and \Rightarrow to denote "changes to" and "logically implies", respectively),

$$(s(t) \rightarrow s(t')) \Rightarrow (M(R)_t \wedge M'(R)_{t'}) \quad (18)$$

$$(s(t) \rightarrow s(t')) \Rightarrow (M(R)_t \wedge M(R)_{t'}) \quad (19)$$

Events depicted in (18) were a model change event and those portrayed in (19) were a model invariance confirmation at two sampling events at time t and t' , respectively. We did require $M(R)_t$ be the most relevant and consistent model for the real time system R at a time t if the predicted system state $\hat{s}(t')$, using this model, at a future time t' was statistically close and similar to the observed system state $s(t')$. For any time instance pair, either (18) or (19) would apply. It was possible that between the sampling events at times t and t' , the system model could change more than once and yet show as though at a later time t' the computed system model remaining the same as the last one. This was perfectly acceptable.

To detect an instance of a model change time point, we partitioned the monitored time length (the observation period) into a set of consecutive epochs $e \in E$. Every epoch began when a model change occurred, and therefore, the length of an epoch – most likely of variable size – was the length of the time duration during which the system model for all computational and controlling purposes remained unchanged. Accordingly, we defined

$$((t \in e) \wedge (t' \in e') \wedge (M(R)_t \wedge M(R)_{t'})) \Rightarrow (e = e') \quad (20)$$

$$(t \in e) \wedge (t' \in e') \wedge (M(R)_t \wedge M'(R)_{t'}) \Rightarrow (e \neq e') \quad (21)$$

As indicated earlier, the duration of an epoch could vary since it was predicated by a model change event. Since the system model remained invariant over an epoch e , we required that the relevant system sampling statistics on system states also remain invariant. If an epoch was large enough we could partition it further into slots of fixed size Δ . Each slot would comprise a bundle of samples on observations capable of yielding relevant system statistics in each slot. Our assumption that Δ was large (large e) was necessary to ensure that we had enough samples to garner appropriate inferences on epoch means, variances, etc. Within an epoch, all distributions were assumed stationary (otherwise, a model change event took place), and therefore, we assumed that the sampling distribution of slot means, slot variances, etc. pointed to epoch means, epoch variances, etc. Obviously such assumptions were ideal; an epoch e need not

be large enough to accommodate large enough slots to render appropriate statistical inferences. But for our model we assumed this to be true. The outline in the next paragraph suggests a way to infer a model change event at a meta-model level at a time point using appropriate CUSUM statistics often used in production planning and quality control.

If the underlying system model did not change, it would be reflected in the time profile of a CUSUM statistics similar to one we would find in a quality or production control environment. For our purpose, we proposed the following CUSUM of order k as defined below. In order to approach our problem comprehensively, we first showed how in a single variable system one could identify model change point. We next extended this to introduce model change point estimation on a bundle of time series.

Consider a single strand of time series x depicting a single variable system. On this, let

$$\begin{aligned} x_i^{s,e}(t) &= \text{ith sample of the observable } x \text{ in slot } s \text{ in epoch } e \\ \mu^e &= \text{slot sample mean of observable } x \text{ in slot } s \text{ in epoch } e \end{aligned}$$

Given these we could define higher order moments of system observables as follows. We defined for a slot s , the k th moment,

$$V_x^{k,s} = \frac{(x_i^{s,e} - \mu^e)^k}{N}$$

where N was some normalization constant in terms of which the average of the k th moment over all slots in an epoch e was the epoch mean of the k th moment, which we assumed to be computationally available at the beginning of our excursion of the epoch e .

$$\mu_e^k = \frac{1}{n_s N} \int_{s=1}^{s=n_s} (x_i^{s,e} - \mu^e)^k ds \quad \text{for } k = 1, 2, \dots, K. \quad (22)$$

We now defined two CUSUM statistics as a function of time

$$cs_+^{s,e} = \sum_i (x_i^{s,e} - \mu^e) \max(x_i^{s,e} - \mu^e - \tau_+, 0) \prod_k (V_x^{k,i} - \mu^e) \max(V_x^{k,i} - \mu^e - \tau_+^k, 0) \quad (23a)$$

$$cs_-^{s,e} = \sum_i (x_i^{s,e} - \mu^e) \min(x_i^{s,e} - \mu^e + \tau_-, 0) \prod_k (V_x^{k,i} - \mu^e) \min(V_x^{k,i} - \mu^e + \tau_-^k, 0) \quad (23b)$$

Notice that the epoch mean μ^e was the same as the sampling slot mean $\mu^{s,e}$ under the assumption we proposed earlier. Also, the sampling means of variance and of other higher order statistics also converge to sampling distribution of epoch statistics. The constants τ_+^m, τ_-^n were appropriate threshold parameters acting as filters for ease of computation.

For our problem, we restricted the CUSUM (23a) and (23b) to a second order product and obtained

$$cs_+^{s,e} = \sum_i (x_i^{s,e} - \mu^e) \delta(x_i^{s,e} - \mu^e) ((x_i^{s,e} - \mu^e)^2 - E(x_i^{s,e} - \mu^e)^2) \delta((x_i^{s,e} - \mu^e)^2 > E(x_i^{s,e} - \mu^e)^2) \quad (24a)$$

$$cs_{-}^{s,e} = \sum_i (x_i^{s,e} - \mu^e) \delta(x_i^{s,e} < \mu^e) ((x_i^{s,e} - \mu^e)^2 - E(x_i^{s,e} - \mu^e)^2) \delta((x_i^{s,e} - \mu^e)^2 < E(x_i^{s,e} - \mu^e)^2) \quad (24b)$$

where $\delta(\vartheta > \varsigma) = \max((\vartheta - \varsigma - \tau), 0)$ and $\delta(\vartheta < \varsigma) = \min((\vartheta - \varsigma + \nu), 0)$.

The constants τ, ν were appropriate threshold parameters. Note that a measure of the k th moment of a distribution across the epoch $E(x_i^{s,e} - \mu^e)^k$ was computed as an average over all exploratory slots before actual monitoring occurs in the rest of the epoch. In other words, at the beginning of every epoch, the first few slots would be used to determine the target statistics for the rest of the epoch. It is against these measures the attendant variable would be monitored.

For convenience, we divided the CUSUM statistics $cs^{s,e}$ by an appropriate normalization constant. A sharp change in the time-derivative of the $cs^{s,e}$ statistics (either of them) portended an end of an epoch and the beginning of the new one, where a new model $M'(R)$ appeared to be the most likely one. Therefore, at a sample point $s(t)$ we computed:

- obtain $d(cs^{s,e})/dt$ at the sampling time t
- $(2/\pi) \text{abs}(\arctan(d(cs^{s,e})/dt)) > \tau_{\text{threshold}}$ implied a model change at the time point t .
- The strength of the hypothesis that there was a model change at time t where (19) was observed was

$$\tau_{\text{threshold}} \quad (\text{between } 0.0 \text{ and } 1.0) \quad (25)$$

The criterion offered two things. On a single-strand mode (system with a single variable time-series), it offered the best estimate of a time-instant when the system underwent a model change, and it offered a measure of belief (on a scale of 0.0 to 1.0) that there was a change point at the time indicated.

Our CUSUM definition in (23) offered us two simultaneous opportunities to detect a model change event. The first term in the product referred to the immediate change as observed in the time series x from the standpoint of the target value, the epoch mean μ^e . The measure here was a short term localized one subject to vagaries of noise. The second term in the product was a measure over a long-range time span (and, hence, less vulnerable to local noise) as the target was compared with the neighborhood variance. Together, our definition rendered a better measure for a model change point: it could detect a model change event if its epoch mean shifted (keeping its variance constant), or its variance shifted (keeping its epoch mean constant), or both.

The single strand decision criterion could be extended to a multitime series decision base. The system state s was a vector with p components, where $p \geq 2$. The component stated $s_i(t)$ and $s_j(t-l)$ could be correlated. Since the complexity of the decision framework in such a case would be horrendous for any real-time system projection, for the time being, we ignored this possibility and assumed that all p dimensions on the state space were mutually independent.

Using the above CUSUM statistics, we could obtain for each component of s the change event point around time t if they at all occurred. Let L be the list of all such events occurring between time t and $t + \delta_\xi$. The time instance t referred to the earliest event time; δ_ξ was the window of opportunity to detect other event points around t . For each model change event registered in the list L we associated a pair of numbers: the time of detection t_{detect}^i and the computed strength of belief $\zeta_i(t_{detect})$ that a model change event was detected on the time series for the state s_i . Within this window δ_ξ , there was a cluster of event points. We proposed that the center of such a cluster (along with its consolidated strength) was the most likely occurrence of a model-change.

The center of an event cluster around t could be computed as follows. Suppose the current cluster coordinate was at $(t_{last}, \zeta_{last}, n_{last})$ where n_{last} was the number of state components clustered at the time point t_{last} with a consolidated strength ζ_{last} . This, along with a single point model change event at some coordinate point $(t_\beta, \zeta_\beta, 1)$ needed to form a new cluster center. The time point of the new cluster was computed recursively as

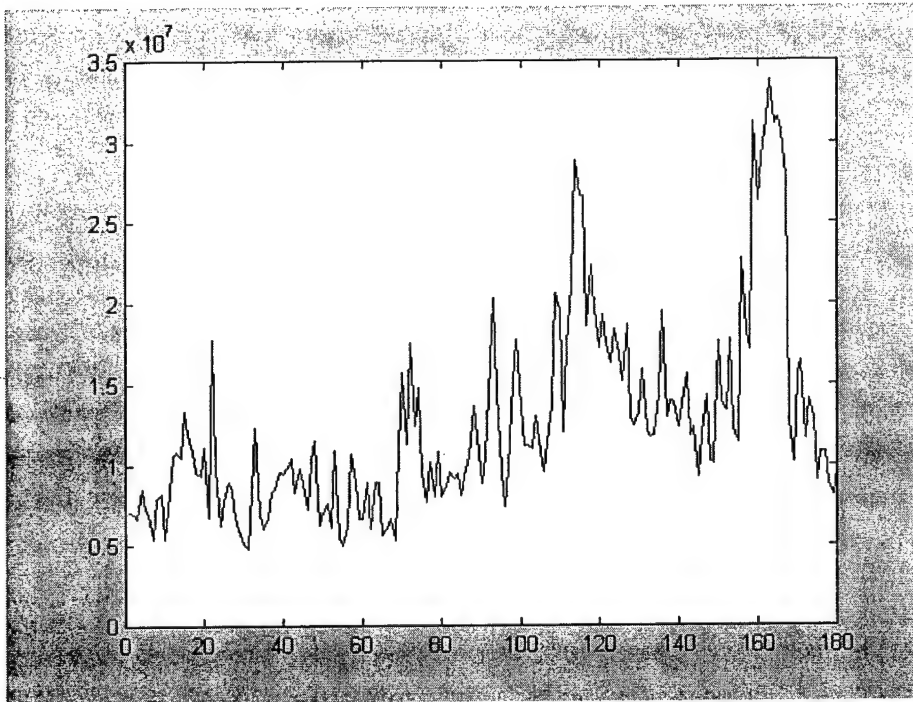
$$t_{new} = \frac{n_{last}t_{last} + t_\beta}{n_{last} + 1} \quad (26a)$$

Similarly, the consolidated possibility index of the new cluster was computed as

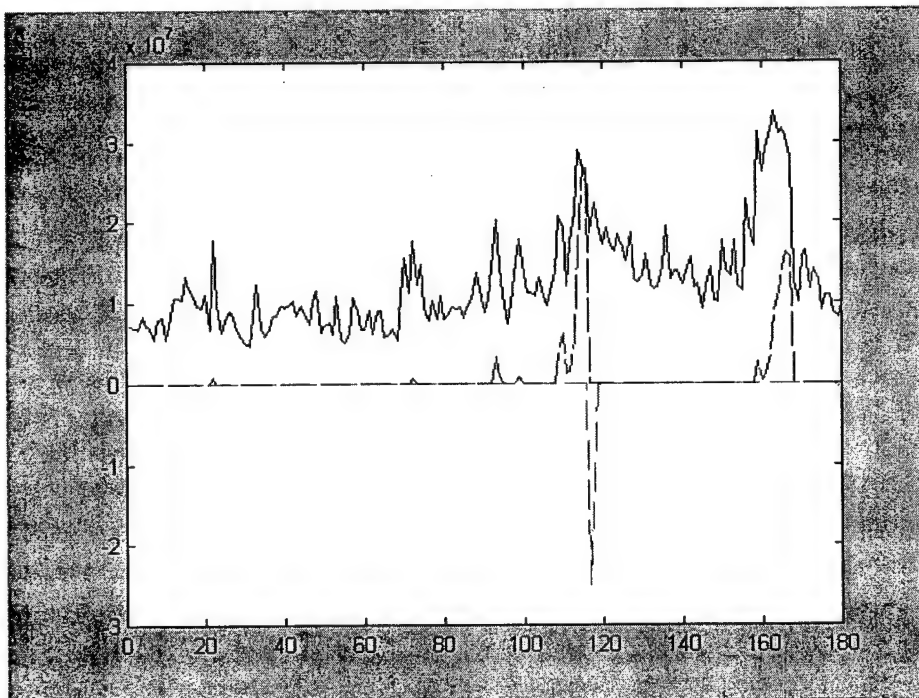
$$\zeta_{new} = \frac{n_{last}\zeta_{last} + \zeta_\beta}{n_{last} + 1} \quad (26b)$$

Notice that $(t \leq t_{last} \leq t + \delta_\xi) \wedge (t \leq t_\beta \leq t + \delta_\xi) \Rightarrow (t \leq t_{new} \leq t + \delta_\xi)$ and $0 \leq \zeta_{new} \leq 1.0$. One might argue that a single strong model change point could be overwhelmed by a large number of weak model change points yielding a false trigger. But, one could also look at it a different way. A single strong showcase (with a possibility value near 1.0) could be construed as a noise particularly when a large number of state parameters were showing a model change at a different time neighborhood even though their individual claim to a model change event was only mediocre.

Using the Matlab computing environment, we first implemented one-sided CUSUM filters for detecting model change points in network traffic data. Exhibit 2, shows the first dimension of the series of Exhibit 1, i.e., number of arriving octets in a router interface, in our traffic data collection environment.

Exhibit 2. Arriving Octets at a Router Interface

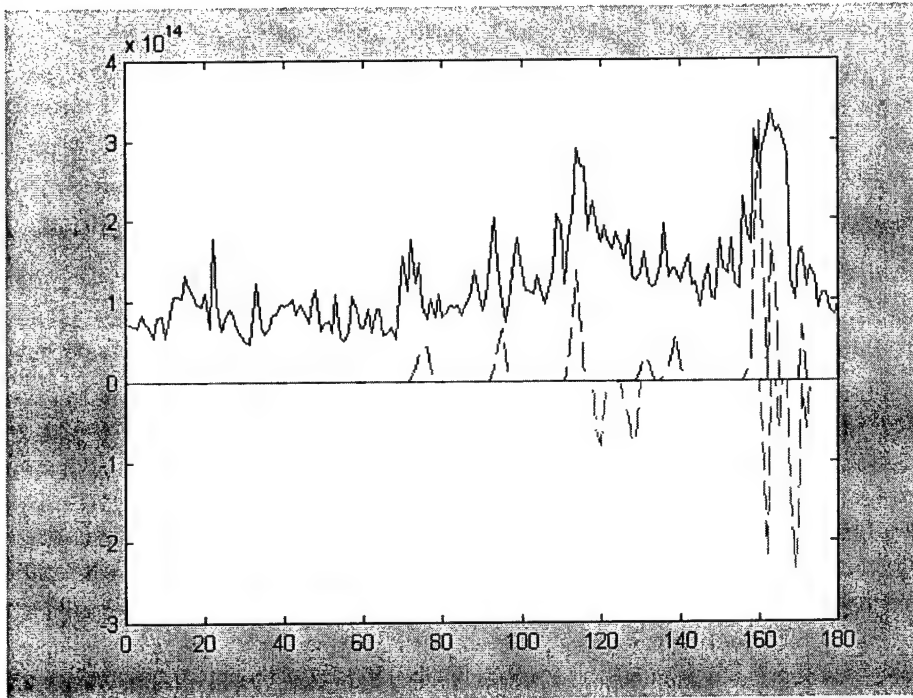
Our CUSUM filters produced the pulses and pulse clusters shown in Exhibit 3 for the upper and lower first-order CUSUMs of the data in Exhibit 2, with the lower CUSUM graphed as its negative, to contrast the two one-sided components.

Exhibit 3. One-sided First-order CUSUM Pulses for Exhibit 2

Cluster centers were computed to $t \cong 115$ and $t \cong 165$.

Filters for second-order CUSUMs, without embedding of the first-order CUSUMs, produced the pulse sets of Exhibit 4 for the same data of Exhibit 2.

Exhibit 4. One-sided Second-order CUSUM Pulses for Exhibit 2

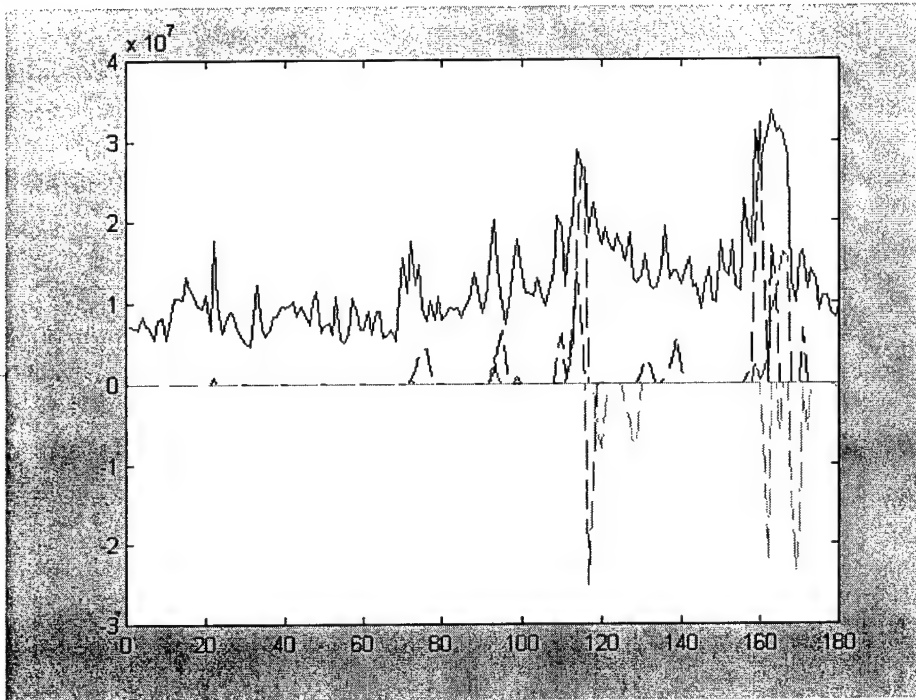


As Exhibit 4 shows, the results for higher-order CUSUMs seem to suffer from additional noise than in the first-order case. Future work should address more stable implementations of higher-order CUSUMs.

A trivial but useful byproduct of the first-order and second-order evaluations above was a heuristic for model change point detection:

- ☐ If both first- and second-order filters signal a change point at τ (computed with the clustering scheme above), then declare a model change point at τ .
- ☐ If only the first- or only the second-order filter signals a change point, then declare a model change point if the pulse amplitude(s) exceed(s) some given threshold θ .

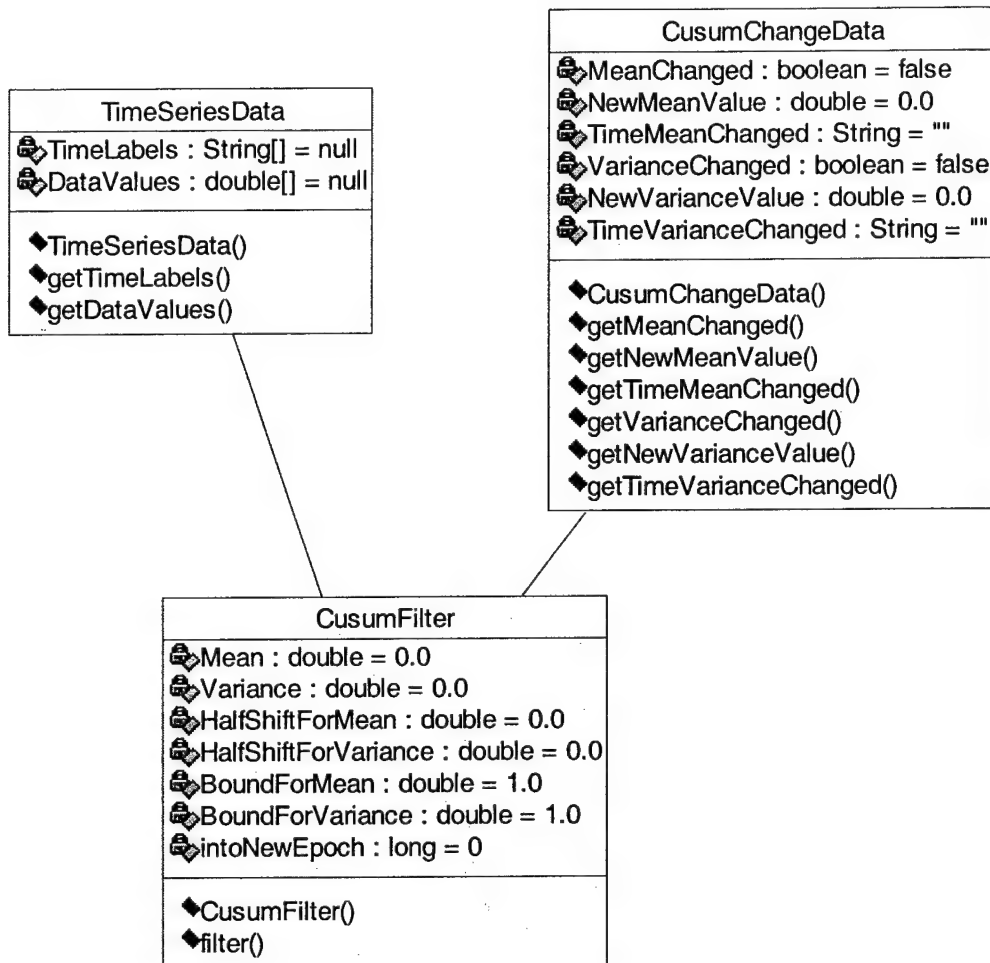
Superimposing the first- and second-order CUSUM results of Exhibits 3 and 4 produces the following graph (Exhibit 5).

Exhibit 5. Combined Clusters from First and Second-order CUSUM Filters

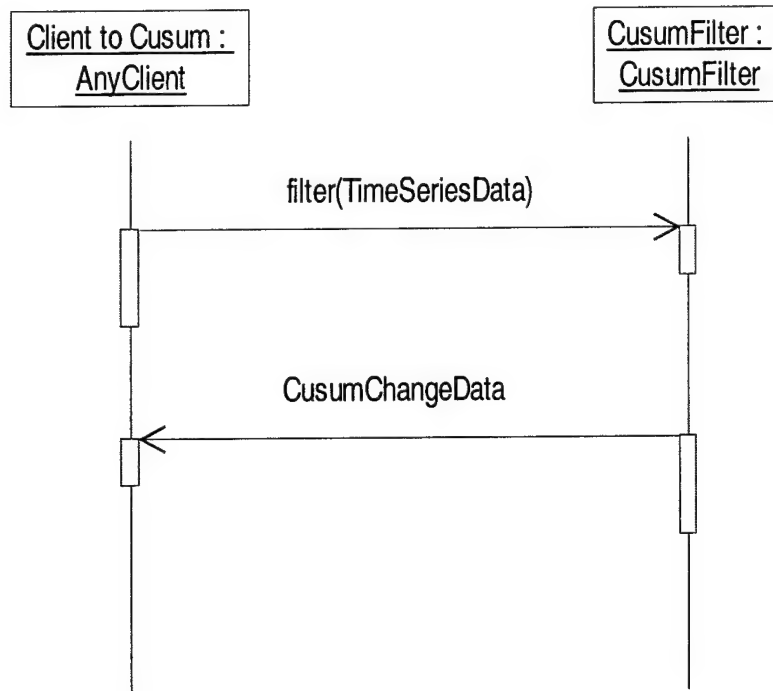
Cluster centers were reconfirmed to hold at $t \cong 115$ and $t \cong 165$.

2.3 TASK 3 – OBJECT MODELING AND SOFTWARE DESIGN**2.3.1 OBJECT MODELING OUR SOFTWARE PROTOTYPE**

The object model of our software prototype had proceeded on two fronts—change point detection and vector autoregression. In regard to change points, after careful examination of the real-time computational costs (under the Java language) of the mathematical models of Tasks 1 and 2, a design and implementation decision was made to consider only CUSUM filter classes in lieu of more general Bayes factors (Exhibit 6).

Exhibit 6. UML Class Diagram for CUSUM Filter

These CUSUM classes were given slices of time series data, and then computed model change points based on mean and variance decision schemes. It was up to the appropriate network management software component to correlate the outputs of the CUSUM filters to the results from other components, to decide the form of feedback for the human user, and to call for any necessary further processing. The Unified Modeling Language (UML) sequence diagram shown in Exhibit 7 could summarize the interaction between CUSUM and its clients (for example, a network manager).

Exhibit 7. UML Interaction Diagram between CUSUM and Its Clients

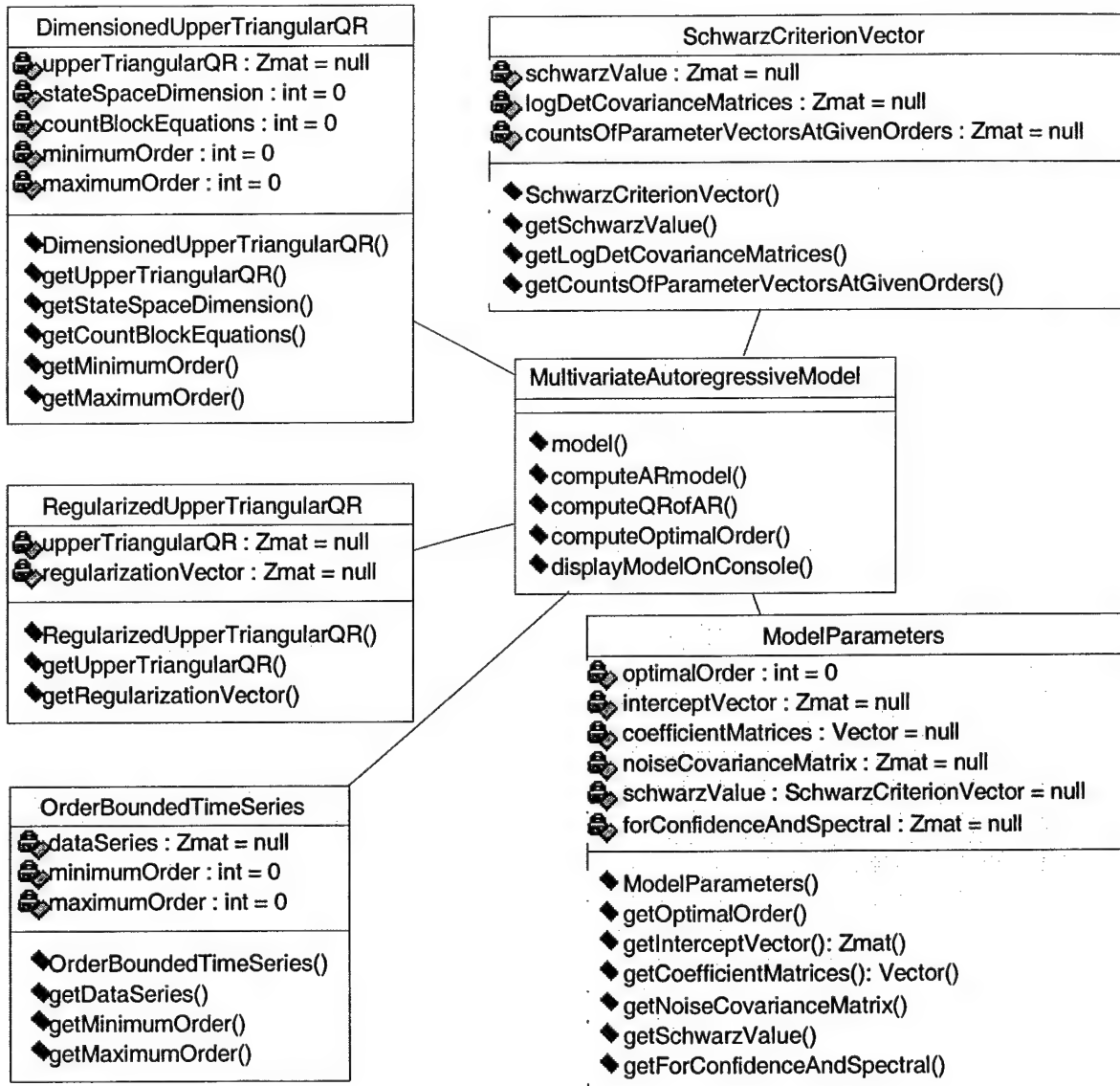
As depicted in Exhibit 6, the CUSUM object model consisted of three main classes, viz. *TimeSeriesData*, *CusumChangeData*, and *CusumFilter*. The first two classes, respectively, encapsulated the input and output to the third class. Both were serializable in the Java sense, so they could, for example, be used as arguments or return types of Java RMI (Remote Method Invocation) methods, if the CUSUM filters were distributed over multiple hosts and devices.

Note that not all instance variables of the *CusumFilter* class are shown in Exhibit 6. For example, our algorithms needed to keep track of the last computed two-sided CUSUM values, for both mean and variance, and also the run lengths for non-zero CUSUM values since the last reset. We kept these data in variables such as *LastMeanUpperCusum* and *nzMeanUpperCusum*. Data and associated collection times were also accumulated in the *CusumFilter* object as it was not known in advance when a change point would occur. We stored the accumulated data in variables *TimeLabelRun* and *DataValueRun*. An integer variable called *ResidualLength* kept track of the data not yet processed, as the *filter()* method of the *CusumFilter* object returns. It was conceivable to rearrange the computations in a slightly different manner, by making the *CusumFilter* object always exhaust the data passed to it, and return a collection of *CusumChangeData* objects. We decided against this second scheme, as we wanted the detection and publishing of change points to occur as soon as possible for real-time sake. Details about the algorithms used by the *CusumFilter* class are given in Section 2.4.3.

Regarding vector autoregression, moving from our original Matlab code to Java required not only objectifying the autoregressive (AR) algorithms, but also selecting appropriate Java numerical packages.

Two packages, Jampack and JAMA, from NIST (National Institute of Standards and Technology) were found extremely useful. We used both in our implementation since in many ways they were supplementary. Jampack provided most of the required functionality but lacked the precision we wanted for Cholesky decomposition algorithms.

Exhibit 8. UML Class Diagram for AR Modeling

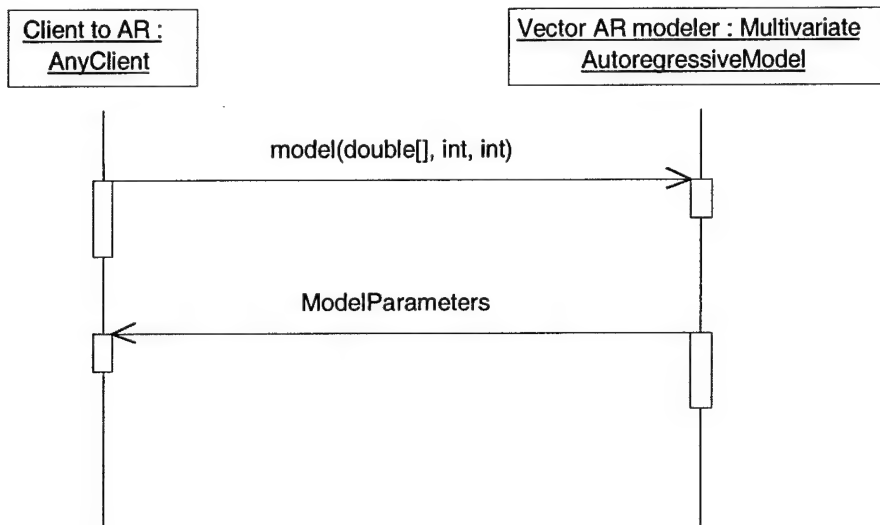


As depicted in Exhibit 8, the object model for vector autoregression consists of six main classes: *OrderBoundedTimeSeries*, *RegularizedUpperTriangularQR*, *DimensionedUpperTriangularQR*, *SchwarzCriterionVector*, *ModelParameters*, and *MultivariateAutoregressiveModel*. The latter class contains the bulk of our algorithms. The first five classes encapsulated intermediate data

that were passed between the methods of the *MultivariateAutoregressiveModel* class, which were made static by design.

Details of the AR model algorithms are given in Section 2.4.3. However, the following items were noted regarding the above object model. First, the following sequence diagram could summarize the interaction between the AR modeler and its clients.

Exhibit 9. UML Interaction Diagram between AR and Its Clients



The arguments passed by a client were the (possibly multivariate) time series, specified as a two-dimensional array, and the minimum and maximum orders desired, passed as integers. Passing a lower and upper bound made the algorithm so flexible as to be applicable to different use case scenarios. For example, if the client had no additional information about the true order of the AR model sought, it would use zero as minimum order and a high enough positive number (i.e., 50) as maximum order. On the other hand, if the client were looking for a linear trend, for example, it would set both minimum and maximum orders to one.

Next, the *Zmat* class played a prominent place in both the object model and the implementation algorithms. *Zmat* is the core class of the Jampack numerical package. It refers to a matrix with *complex* entries, with methods for manipulating submatrices. The design of Jampack calls for familiar matrix operations to be implemented in *suites*, which are simply Java classes. For example, the LU decomposition was done with the *Zludpp* suite, more precisely in the constructor of the *Zludpp* class, with the *Zmat* object being LU-decomposed as argument.

Next, the methods of the *MultivariateAutoregressiveModel* class were all static, in the Java sense. This was in accordance with the general philosophy behind the Jampack package itself. This technique saved us from passing heavy objects around our algorithms.

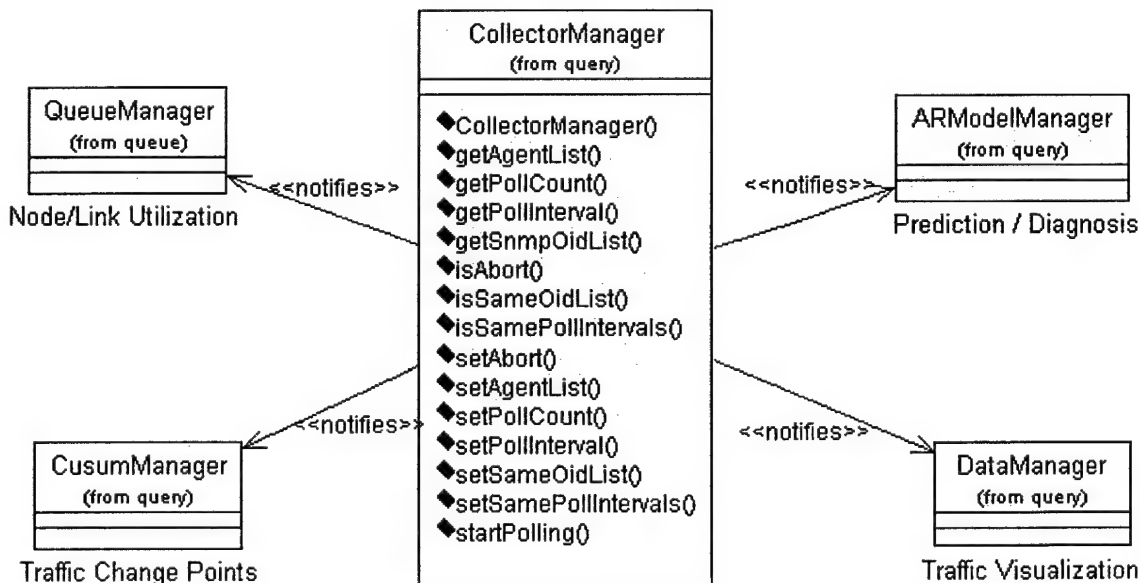
Finally, the *displayModelOnConsole()* method of *MultivariateAutoregressiveModel* was just a trivial example for showing the calculated parameters of a vector AR model. It was actually up to the clients of the class to decide how to display these parameters.

2.3.2 GRAPHICAL USER INTERFACE

The graphical user interface (GUI) underwent many changes. The GUI provides not only the point and click interface that all users are accustomed to, but also a graphical depiction of some of the more complicated aspects of network management and network modeling. All of this information is obviously related to traffic on the network. Since the raw traffic information is required at almost every level of the RTNM, it made sense to design an architecture with a central data collector.

The CollectorManager is, as it would seem, responsible for the collection of traffic data. With these data we could determine things such as the number of packets in, packets out, packets dropped, packets per second, and so on. The raw data were collected by the CollectorManager, which passed this information to each of the objects responsible for calculating the information we needed (see Exhibit 10).

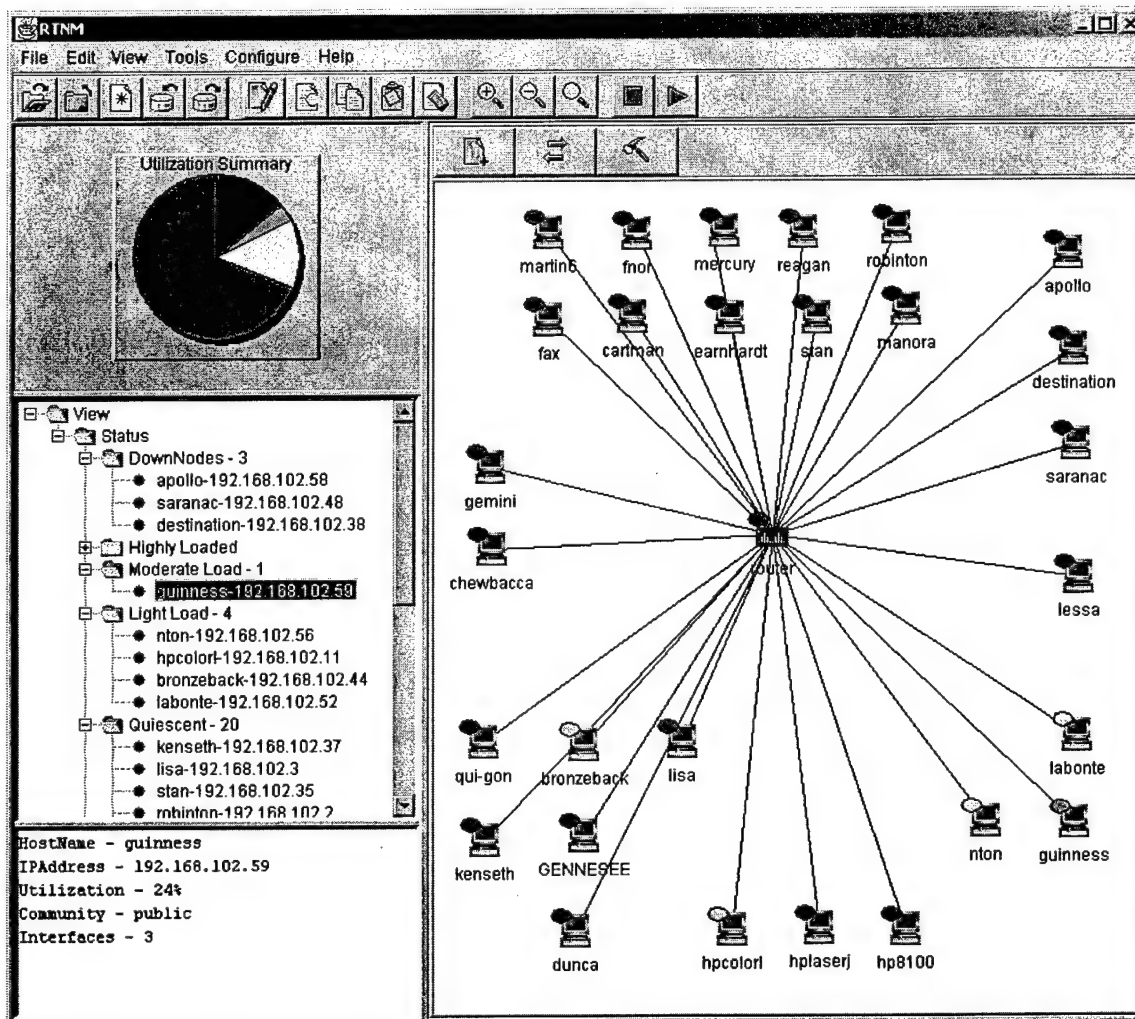
Exhibit 10. RTNM Data Collection Overview



These objects were responsible for all the information generated by the RTNM, including network topology, traffic, modeling and prediction, and access to common network management tools. The network view provided a “top level” view of the network or a region of the network (Exhibit 11). This allowed the network administrator to see the status of the network at a glance. Changing colors indicated node and link utilization. A pie chart representing the current region

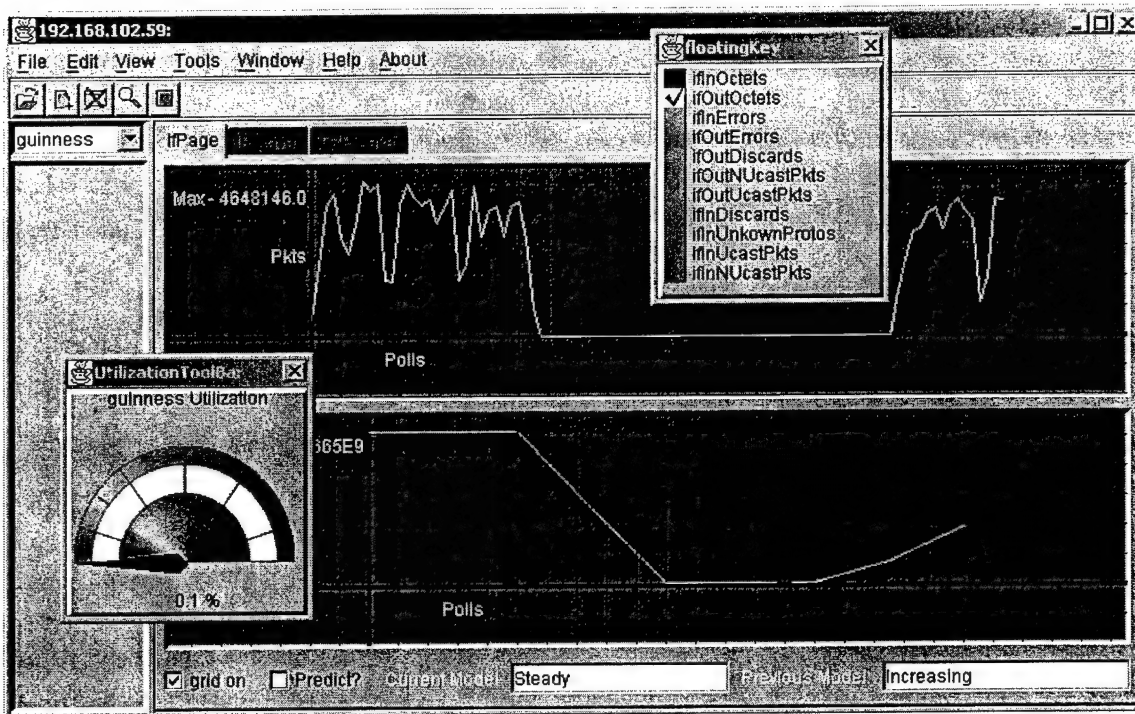
indicated the relative levels of utilization for this view (low, light, medium, high, down, and unmanageable).

Exhibit 11. The Network View



By clicking on a specific node, one could drill down and retrieve more specific information about that node. This drill down view presented the user with what we call the node monitor view (Exhibit 12). The node monitor view provided a traffic level view of a particular node on the network. This would usually be a switch or router, although it could be a workstation acting as a router or server. At this level, one could inspect the actual number of packets going in and out of this node. The traffic is shown for both the interface and Internet protocol (IP) layer.

This view also shows a graph of the CUSUM model used in the determination of network traffic change points.

Exhibit 12. The Node Monitor

Although the RTNM program provides this specialized network information, no network management package would be complete without some basic network troubleshooting tools. The RTNM provides a database browser, which provides a simple way of monitoring and changing information being stored about the network. This means the users need not concern themselves with database management or SQL statements (see Exhibit 13).

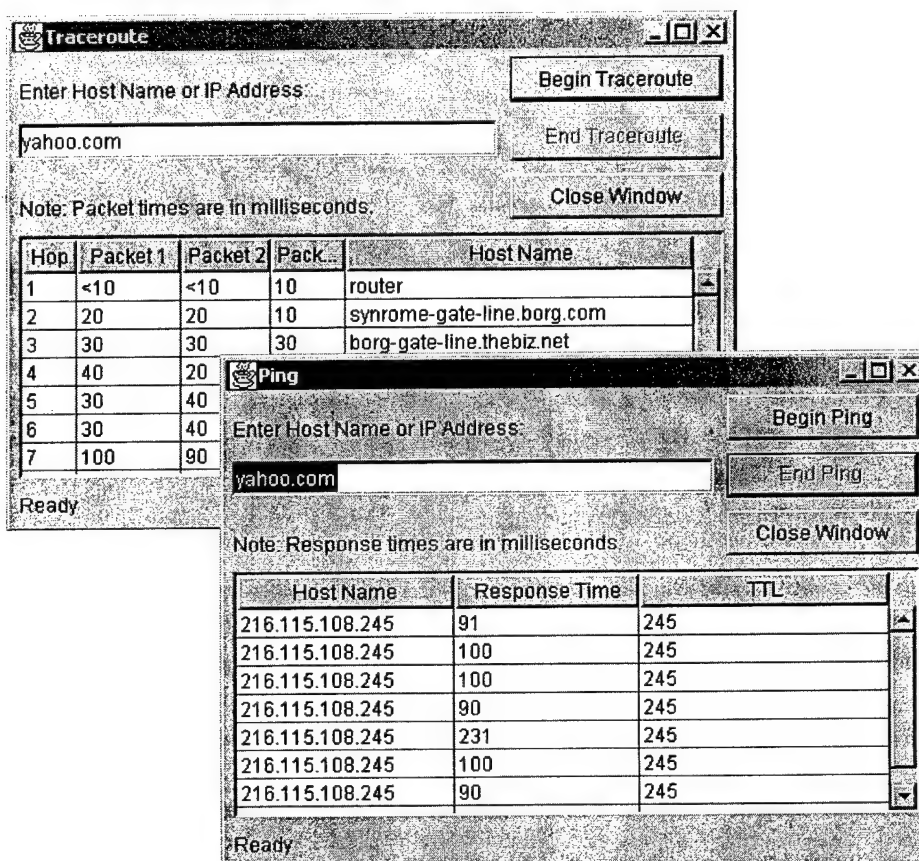
Exhibit 13. Database Browser

The screenshot shows the 'Database Browser' application window. The title bar is 'Database Browser'. The menu bar includes 'File'. The window contains fields for 'URL' (jdbc:mysql://pc6/snm), 'Driver' (org.gjt.mm.mysql.Driver), 'User' (rtm), and 'Password' (masked with asterisks). Below these fields is a toolbar with icons for file operations. On the left is a list of database tables: downNodes, ifInOctets, ifInfo, ipInfo, knownNodes, links, nodes, startNodes, sysInfo, and trap. The 'ifInfo' table is selected, and its fields are displayed in a table:

Field	Type	Null	Key	Default	Extra
ipAddress	varchar(15)		PRI		
readCommunity	varchar(30)	YES			
ifIndex	int(11)		PRI	0	
ifDescr	varchar(100)	YES			
ifType	int(11)	YES			
isVirtual	char(1)	YES			
ifMtu	int(11)	YES			
ifSpeed	int(11)	YES			
ifAdminStatus	int(1)	YES			
ifOperStatus	int(1)	YES			

The RTNM package also provides access to the Ping and Traceroute tools, which are considered standard troubleshooting mechanisms. Our tools have a friendlier graphical interface added to them. This provides ease of use and eliminates the need to log into a separate shell in order to run these commands, as shown in Exhibit 14.

Exhibit 14. Screenshot of Ping and Traceroute Tools



2.3.3 SNMP NETWORK MANAGEMENT WITH JDMK

JDMK (Java Dynamic Management Kit) is being used as a foundation for SNMP support. JMAPI (Java Management Application Programmer's Interface) was the original API that was going to be used on this effort. However, Sun dropped JMAPI and started working with JMX (Java Management eXtension). An evaluation of other management Java APIs for network management was completed (see Table 8). Because JDMK supported Sun's new JMX effort, we used JDMK.

Table 8. Evaluation of Management Java APIs

Vendor	Product	Version	Platform	Description	Reqmnts	Pros	Cons	Cost	Comment
AdventNet	SNMP	3.1	All/Java	SNMP management software	Provide tools for SNMP information gathering and presentation	Free; Java-based	Buggy; unable to handle partial SNMP responses	Free/Trial Version	Excellent program, but not stable enough yet
Castle Rock Computing	SNMPC	5.0	98/NT	Distributed SNMP management support	Provide tools for SNMP information gathering and presentation	Good network visualization	Poor network discovery	Evaluation: Free Enterprise: \$2700	
Sun Microsystems	Java Dynamic Management Kit	4.0	NT	Distributed SNMP management support	Provide tools for SNMP information gathering and presentation	Widely used; adheres to JMX spec	Seems to be a bug with Cisco routers	\$6000	Purchased this version; awaiting upgrade to 4.1
Hewlett Packard	Open View	Unknown	Windows/Solaris	Distributed network management	SNMP Management	Widely used and accepted	Expensive	\$10,000 to \$20,000	This is really more of a standard with which to compare
Sun Microsystems	JMAPI	4.0	All/Java	SNMP management software	Provide tools for SNMP information gathering and presentation	JAVA	Immature	??	This program not purchased in favor of the JMX framework
Sun Microsystems	JMX (Java Management Extension)	Beta	All/Java	SNMP management software	Provide tools for SNMP information gathering and presentation	Will provide a level of standardization to network mgmt	Not yet fully designed	FREE	We are using a version of this which was included in the JDMK
Naval Research Lab	VENoM- (Virtual Environment for Network Monitoring)	Beta	UNIX	3-D Virtual Network Manager	Heavy duty processing power	Easy to use and understand	3-D is too complex for Java	Unknown	Looks pretty, but not very practical

The JDMK supports SNMP v1 and SNMP v2 (including bulk requests). Using the SNMP Manager API provided with JDMK we were able to:

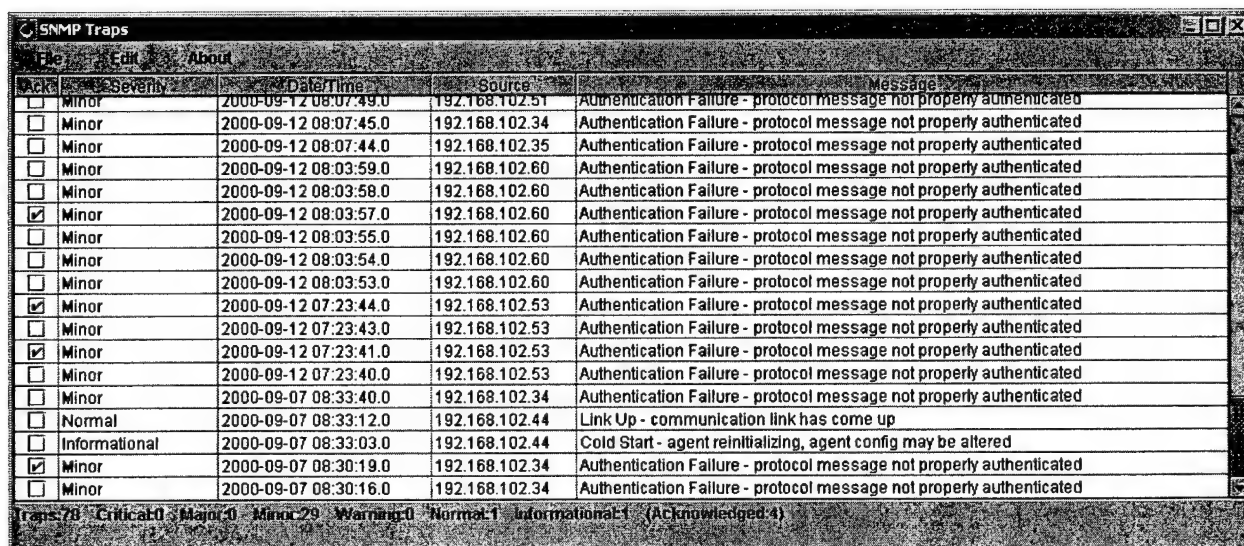
- ☐ Send SNMP requests (and receive responses)
- ☐ Receive SNMP traps
- ☐ Query SNMP nodes in synchronous and asynchronous modes
- ☐ Translate OIDs (Object Identification numbers) (using the dictionary produced by mibgen)

This API and its functionality provided the Real Time Network Manager with fuller network management support, more specifically more robust SNMP support.

Our prototype created for Phase I of the SBIR included the ability to send SNMP requests and receive their responses. This proof of concept only supported SNMP V1. The current version of JDMK 4.2 supports SNMP V1 and V2c and allows the SBIR Phase II application, RTNM, to support a wider range of nodes with the addition of SNMP V2c.

JDMK provides support for SNMP Traps. By receiving SNMP Traps RTNM can become a true network management system. Currently RTNM can collect, store, and display traps sent from correctly configured nodes (Exhibit 15). This information may be used as an input to our expert system to inform the network administrator of problems that may be occurring.

Exhibit 15. The SNMP Trap Browser



Ack	Severity	Date/Time	Source	Message
<input type="checkbox"/>	Minor	2000-09-12 08:07:49.0	192.168.102.51	Authentication Failure - protocol message not properly authenticated
<input type="checkbox"/>	Minor	2000-09-12 08:07:45.0	192.168.102.34	Authentication Failure - protocol message not properly authenticated
<input type="checkbox"/>	Minor	2000-09-12 08:07:44.0	192.168.102.35	Authentication Failure - protocol message not properly authenticated
<input type="checkbox"/>	Minor	2000-09-12 08:03:59.0	192.168.102.60	Authentication Failure - protocol message not properly authenticated
<input type="checkbox"/>	Minor	2000-09-12 08:03:58.0	192.168.102.60	Authentication Failure - protocol message not properly authenticated
<input checked="" type="checkbox"/>	Minor	2000-09-12 08:03:57.0	192.168.102.60	Authentication Failure - protocol message not properly authenticated
<input type="checkbox"/>	Minor	2000-09-12 08:03:55.0	192.168.102.60	Authentication Failure - protocol message not properly authenticated
<input type="checkbox"/>	Minor	2000-09-12 08:03:54.0	192.168.102.60	Authentication Failure - protocol message not properly authenticated
<input type="checkbox"/>	Minor	2000-09-12 08:03:53.0	192.168.102.60	Authentication Failure - protocol message not properly authenticated
<input checked="" type="checkbox"/>	Minor	2000-09-12 07:23:44.0	192.168.102.53	Authentication Failure - protocol message not properly authenticated
<input type="checkbox"/>	Minor	2000-09-12 07:23:43.0	192.168.102.53	Authentication Failure - protocol message not properly authenticated
<input checked="" type="checkbox"/>	Minor	2000-09-12 07:23:41.0	192.168.102.53	Authentication Failure - protocol message not properly authenticated
<input type="checkbox"/>	Minor	2000-09-12 07:23:40.0	192.168.102.53	Authentication Failure - protocol message not properly authenticated
<input type="checkbox"/>	Minor	2000-09-07 08:33:40.0	192.168.102.34	Authentication Failure - protocol message not properly authenticated
<input type="checkbox"/>	Normal	2000-09-07 08:33:12.0	192.168.102.44	Link Up - communication link has come up
<input type="checkbox"/>	Informational	2000-09-07 08:33:03.0	192.168.102.44	Cold Start - agent reinitializing, agent config may be altered
<input checked="" type="checkbox"/>	Minor	2000-09-07 08:30:19.0	192.168.102.34	Authentication Failure - protocol message not properly authenticated
<input type="checkbox"/>	Minor	2000-09-07 08:30:16.0	192.168.102.34	Authentication Failure - protocol message not properly authenticated

Traps: 78 Critical: 0 Major: 0 Minor: 29 Warning: 0 Normal: 1 Informational: 1 (Acknowledged: 4)

RTNM has the ability to query multiple nodes at once with the addition of asynchronous communications. Our Phase I prototype implementation used synchronous querying of nodes as JMAPI support was limited. This process allowed RTNM to query a single node at a time,

prohibiting a true picture of the entire network at once. Since we were using JDMK we could query multiple systems at once, thereby making our query process run concurrently and allowing our displays and data to be a true snapshot of the network.

We used information from MIB II (defined in RFC 1213) that contained many OIDs. Working with OIDs can be very confusing. For example, the variable `ifInOctets` was OID 1.3.6.1.2.1.2.2.1.10. JDMK translated the OIDs into useful variables so one could just ask for `ifInOctets` for interface 1 as easily as using `ifInOctets.1`. This provided an easy way to read and write, as well as to debug code. JDMK provided this ability to translate OIDs.

These added features provided a fuller support of SNMP in RTNM. RTNM has the ability to collect all physical and network layer OIDs at defined time intervals. This transpired because JDMK supplied RTNM with a robust API for gathering SNMP information about a network.

2.3.4 MODEL BASED DIAGNOSTICS

Problem diagnosis in any application domain, in particular network management, is an intractable problem in general. A given observed symptom may have been produced by a myriad of root causes. In practice, we usually proceeded by iteration, starting off with a collection Ψ of the most likely causes. Then for each iteration, we removed as many unlikely causes from Ψ as possible, through appropriate (usually stochastic) elimination techniques. We were looking at integrating vector autoregressive modeling with two such techniques, namely Bayesian belief inference and vector time series causality tests.

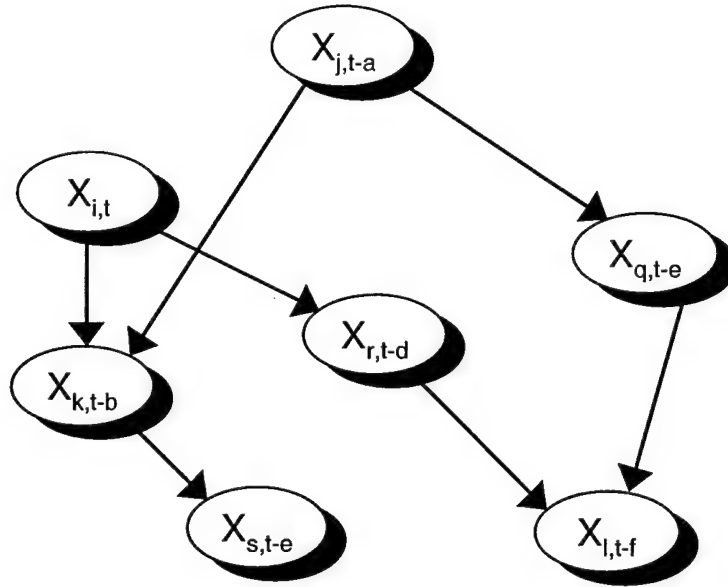
For a given pair (S,M), where S is a set of nodes (routers or hosts) and M is a set of Management Information Base (MIB) variables pertaining to S, a Bayesian belief network, such as the one shown in Exhibit 16, could be constructed. The nodes in the network refer to the possibly lagged variables of M. There is an arc $X_{i,t-a} \rightarrow X_{j,t-b}$ when it is believed that $X_{i,t-a}$ significantly causes $X_{j,t-b}$ directly, that is, when the conditional probabilities

$$p(X_{j,t-b} = v \mid X_{i,t-a} = w)$$

are deemed non-negligible. In our current thinking, data mining techniques could be used to build the belief network. First, the possible values of each $X_{i,t-c}$ should be quantified to a small number of levels v . Then, a counting of observed values could be performed to produce

$$p(X_{j,t-b} = v \mid X_{i,t-a} = w) = \frac{|X_{j,t-b} = v \cap X_{i,t-a} = w|}{|X_{i,t-a} = w|}$$

Once a belief network was built, it could be used to collect potential causes of an observed value $X_{j,t-b} = v$ through standard Bayesian inference techniques. The main issue here was the large number of lagged $X_{i,t-a}$ and the intractability of Bayesian network inference (except in very special cases, such as single connectedness).

Exhibit 16. Bayesian Belief Network

Operational statistical causality or non-causality tests between two or more variables of vector AR models were also researched. C.W.J. Granger introduced in 1969 the basic framework for non-causality tests in the linear case. This framework has since been extended in various directions, including non-linear and non-parametric models. In the case of network management, the goal was to devise reliable tests that permitted the elimination of as many variables X_i as possible as causes of the observed value of another variable X_j . At the present time, network managers manually conduct similar "tests" ad hoc. It is desirable to assume as little as possible about the "functional" relationships between variables as such relationships are extremely difficult to ascertain in the case of networking, and in fact may be said to be the problem itself. In our work, we also imposed the additional constraint that the tests were easy and efficient to compute, as our goal was to use them in real-time monitoring scenarios. Now, the k th component of the vector AR equation

$$X_t = W + \sum_{i=1}^p A_i X_{t-i} + \epsilon_t$$

may be written as follows, with the obvious meaning for the indices

$$X_{k,t} = W_k + \sum_{i=1}^p \left(\sum_{j=1}^n A_{i,k,j} X_{j,t-i} \right) + \epsilon_{k,t}$$

Hence, the null hypothesis that $X_{r,t}$ does not linearly cause $X_{s,t+}$ is

$$H_0 : A_{1,s,r} = 0, A_{2,s,r} = 0, \dots, A_{p,s,r} = 0$$

Of course, the causation relationship between two variables, if any, need not be linear at all. In the linear case for two variables only, we designed a procedure consisting of two regressions and

the computation of two statistics with F-distribution and χ^2 -distribution, respectively, under hypothesis H_0 . In the general case, we attempted to imitate the linear case, using Taylor series expansion. The procedure and the obtained results require further study.

We investigated a possible use of Fourier descriptors from the vector AR model equation

$$X_t = W + \sum_{i=1}^p A_i X_{t-i} + \varepsilon_t \quad (27)$$

The vector X_t of equation (27) may be viewed as the parametric equations of a hypercurve in dimension m . Information was extracted from this hypercurve through appropriate functions defined over its points. The first function of interest to us was the *curvature* $\kappa(t)$, which is the magnitude of the rate of change of the tangent vector with respect to the arc length. That is, if

$$T = \frac{dX_t}{dt}, \quad ds = \sqrt{dX_t \cdot dX_t}$$

then,

$$\kappa(t) = \left\| \frac{dT}{ds} \right\| = \left\| \frac{\frac{dT}{dt}}{\frac{ds}{dt}} \right\|$$

Fourier descriptors were obtained as follows. First, samples of $\kappa(t)$ were transformed through a Fast Fourier Transform (FFT). Second, as X_t was real, the Fourier coefficients were identical in both the positive and negative frequency axes; hence descriptors were obtained by dividing the absolute value of each positive frequency coefficient by the absolute value of the DC (direct current) coefficient.

Autoregression descriptors could also be obtained for $\kappa(t)$. Applying equation (27) to the curvature, we obtained the following formulation

$$\kappa(t) = w + \sum_{i=1}^h a_i \kappa(t-i) + \sqrt{\beta} \alpha_t$$

where the residual was written in the more convenient form $\sqrt{\beta} \alpha_t$, which is possible because of the single dimension. We could then take the sequence $(a_1, a_2, \dots, a_h, \frac{w}{\sqrt{\beta}})$ as feature vector. In

many particular cases there were other possible curve functions in addition to the curvature. For example, if $m = 2$, we could consider the centroid Ω of the shape obtained by "completing" the curve X_t by symmetry around the line joining its end points. The function $\rho(t)$ given by

$$\rho(t) = \text{distance between } \Omega \text{ and } X_t$$

then produced feature vectors via FFT and autoregression.

Likewise, still in the case $m = 2$, the two components of X_t could be viewed as the real and imaginary parts of a complex number z . The latter could be Fourier transformed directly. Here, the DC component could be dropped, but the coefficients in the negative frequency axes were needed. The sequence of non-DC Fourier coefficients scaled by the absolute value of the first non-zero non-DC coefficient produced the feature vector. Our experiments showed us that the curvature function seemed to have the best performance.

2.3.5 FUTURE DIAGNOSTICS INPUT

The implemented vector autoregressive modeling in Sections 2.3.1, 2.4.3, and 2.4.4 could form a basis for prediction and diagnosis capabilities for our software prototype. Several interrelated research areas were explored regarding the significance of the vector AR modeling.

First, a vector AR model could be used as a point in an operating space. More precisely, for any entity E to be monitored (i.e., a single device, a subregion of a network, or an entire network), we could define an associated *topological space* O_E , where each point represented a state in the life cycle of entity E .

A *serialized* AR model was an obvious candidate for implementing such a point. With the notation of our previous report, if the current AR model of entity E is given by

$$X_t = W + \sum_{i=1}^p A_i X_{t-i} + \varepsilon_t \quad (28)$$

then, by adopting a simple row/column serialization scheme (row 1, followed by row 2, followed by row 3, etc.) for the model vectors W , C , and A_i (where C is the covariance matrix for the noise component ε_t), a numerical representation of the state of E was obtained. There is an obvious *metric* on the space O_E , namely the *distance* between the two serialized representing vectors, which is the *norm* of their difference.

There was an obvious issue regarding the above proposal, viz. the vector AR model need not always be of the same order p , throughout the life cycle of entity E . Indeed, if the order was always the same, then the topological space O_E would in fact be a *normed vector space*. A practical but not necessarily satisfactory solution was to force the order to be always the same. For example, if we were interested in trends in the monitoring activities, we could, as a first approximation, assume order 1. The design of our vector AR modeler indeed permitted such a choice. A more formal approach defined a *metric dist* on O_E as follows.

A point $x \in O_E$ was viewed as a pair (p, \underline{X}) , where p was the order of the vector AR model corresponding to x , and \underline{X} was the serialized representation of this AR model, as described above.

Then, for any points $x = (p, \underline{X})$, $x' = (p', \underline{X}')$ and $x'' = (p'', \underline{X}'')$ in O_E , define

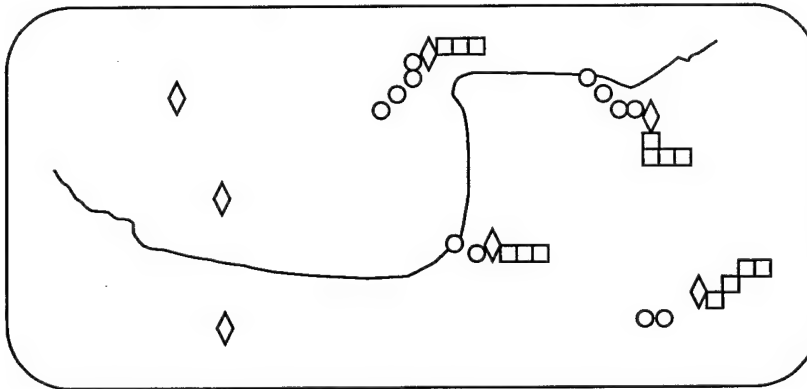
$$\text{dist}(x, x') = \infty \quad \text{if } p \neq p'$$

$$\text{dist}(x, x') = || \underline{X} - \underline{X}' || \text{ if } p = p'$$

where the vector norm could be any norm, since it was well known that on a finite-dimensional vector space, all norms were topologically equivalent. In an actual software implementation, we would of course select a computationally inexpensive norm. It was a straightforward matter to check that *dist* was a valid metric, in the mathematical sense, namely that it was symmetric, satisfied the triangular inequality, and for any $x \in O_E$, x was the only point whose distance to x was null.

Practical values of the topological space O_E would emerge only if we could attach network management semantics to the points of the space O_E . First, the lifecycle of entity E became a *trajectory* within the space O_E . In Exhibit 17, we show this lifecycle trajectory as a continuous line. Second, a point may represent a *normal behavior* of entity E , or a *pre-fault behavior* (depicted as an oval in Exhibit 17), a *fault behavior* (depicted as a diamond), or a *post-fault behavior* (depicted as a rectangle). Then, for example, if during real-time monitoring, the state being observed was close to that of a pre-fault or fault behavior, according to the above metric, appropriate alarms might be thrown. The research problem here was the characterization of a vector AR model in terms of faults or precursors to faults. An experimental task we conducted was the modeling of the same network at the same time of the day, for several days.

Exhibit 17. Operating Space as a Topological Space

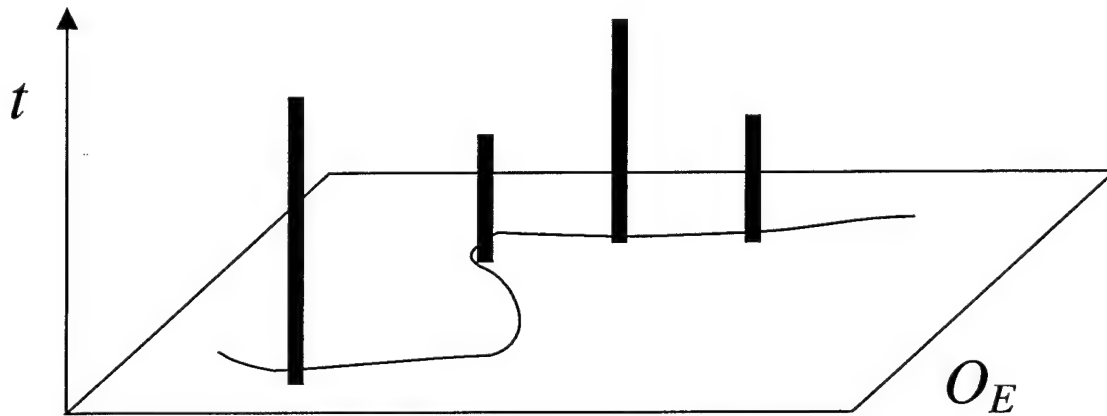


We noted that the space O_E was a compressed view (more accurately, a flattened view) of the life of a managed entity. Indeed, the vector AR model did not tell the entire story. As a trivial example, consider the case of a one-dimensional model $x = 0.08 + 0.12 t$ for the number of packets dropped by entity E . For small values for time t , the value of x probably signified nothing. However, for large values of t , trouble-indicating thresholds would be exceeded. The only thing we could say is that the model was a precursor to trouble, as it was an increasing function of t , for a quantity that could not grow indefinitely. Whether it actually signified fault, or closeness to fault, must be ascertained by the value provided by x on the time dimension.

In general, the extra time dimensions, corresponding to evolution in time of the same model, were given by X_t in equation (28), or more precisely, the columns of X_t . Our software prototype was envisioned to have to watch both the space O_E for change of model, and the time dimensions for the evolution of the same model (Exhibit 18). The black line segments emanating from the

O_E space represent levels of a state space variable (i.e., a column of X_t). Of course, when the state space dimension was larger than 1, we could not really provide a three-dimensional depiction of the lifecycle of the managed entity E . Diagnosis through the vector AR models might also be enhanced with rules and stochastic causal analysis. Work in this area should be continued.

Exhibit 18. Topological Operating Space with Time Dimension



2.3.6 STUDY OF MOBILE AGENTS

The two well-known standards for agent-based computing, OMG Mobile Agent Software (MASIF) and Foundation for Intelligent Physical Agents (FIPA), were considered for their relevance to network monitoring in general, and to our software prototype in particular. We looked at two FIPA implementations, Jade from CSELT of Torino, Italy, and FIPA-OS from Nortel Networks of Ontario, Canada. We also considered more general frameworks that subsumed both MASIF and FIPA. In this regard, we experimented with the Grasshopper system from IKV++ GmbH, Germany, which permits both MASIF and FIPA plug-ins.

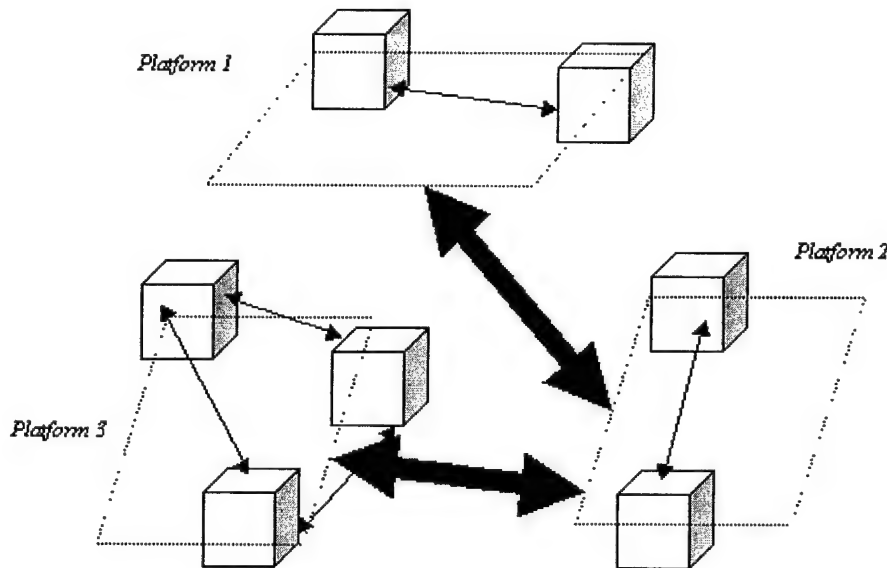
After some preliminary experimental work, we decided not to pursue the FIPA-OS route but to retain the Jade system. The main reason was that we were not able to exercise agents living on multiple platforms using FIPA-OS. The Jade system appeared suitable for two main concerns of network monitoring and management: (1) the large amount of network traffic data, and (2) the heterogeneity of the network operating environment.

Regarding the second issue, Jade permitted devices and machines running different operating systems to host interoperable agents, as long as they could run a Java virtual machine, as Jade is entirely written in Java. Jade, like any FIPA-compliant agent system, allows a flexible logical partitioning of cooperating agents, within a network management domain.

In Exhibit 19, we show this partitioning, the cubes representing entities to manage and to be managed. The dashed parallelograms are what FIPA and Jade call agent platforms, the full skinny arrows represent Java Remote Method Invocation (RMI) communication between agents

belonging to the same platform, and the fat arrows signify CORBA IIOP communication between agents contained in separate platforms. This division scheme then provides different levels of granularity, in terms of functionality and management requirements.

Exhibit 19. Jade Platforms for Network Management



A FIPA agent, i.e., a Jade agent, always executed within an agent container, which in Jade is a Java virtual machine. An agent platform was made of one or more containers, which could span multiple devices and hosts, as shown in Exhibit 19. We could roughly map the concept of platform to that of a management domain. A benefit of this mapping followed from the use of Java RMI as object communication framework, in lieu of the more expensive language-independent CORBA IIOP. An enterprise network would consist of one or more agent platforms.

We experimented with the Jade FIPA and the Grasshopper agent systems. Agent systems allow various nodes on the network to share in the monitoring and management of the network, as opposed to a central management station querying each node and making all decisions. We concentrated on the FIPA and MASIF plug-ins in Grasshopper. Although the Grasshopper agent system was very attractive for general agent computing, we felt that it was too demanding in resources for our purposes. We did not, therefore, pursue it further, and returned to the Jade FIPA system. These systems, however, were agents and therefore they had to be installed on network devices. Unfortunately, even though these types of agents would be extremely useful, the reality was that most network managers were reluctant to install software on import network devices.

2.3.7 NETWORK MANAGEMENT LOAD DISTRIBUTION

Management load across a network that is supposed to be managed and controlled in real time is both spatial and temporal. Local observables include all that pertain to a given node i.e., traffic in and out of the node, the server's rate at that node, the effective buffer volume available at that node, the AR and truncated trend line features of state variables at that node, the model-change time point computations, and model characteristics at that node. Also, at each node, appropriate agents at that node would carry out relevant regional computations. If a radius of k-hops was construed to be the maximum spread of a manageable region, then for all region-definitions from 1 to k hops (both inclusive) we would have to compute regional data at that node.

For computational ease, the entire manageable region within h-hops from a super-manager node (or the controller node) could be arranged in a hierarchical fashion. If a d-depth tree was sought, at each level except for the leaf nodes, a regional controller would be defined with roughly $N^{1/d}$ subregions under its control. Management load would be distributed over regional and subregional controllers.

For our purpose, we configured the system with a finite number of non-overlapping regions each of which was roughly within some k-hops radius from the regional controller. All these controllers periodically sent their computed regional and local data to the network super-controller (the observer). It was incumbent upon the latter to report functionality of the network to its clients particularly when the network's traffic rate changed, congestion (local and/or distant) developed, or improved to a normalcy.

For a node, not only did its network neighborhood need to be ascertained but its temporal dimension had to be addressed as well. How long should a node retain its data (raw and computed)? How long should the regional data be stored at a node? Our preliminary finding attempted to conceptualize it in the following way. In a dynamic and an adaptive system, the measurement of system states takes a crucial dimension since the window of opportunity to amend a measurement may often be narrow. For a system that remains invariant in time, one has an infinitely long window to improve upon measurements before it is reported. In our system, we noted that some local state information at a time t might be required for the following reasons:

- ☐ To project a measure for the same state at a later time $t + \Delta$
- ☐ To project a measure for a regional state (region being the network neighborhood within k hops) at a future time $t + \Phi$, i.e., regional congestion level measures, etc.

Therefore, a local system state θ_i at a node had to be available for at least $T_i = \max(\Delta_i, \Phi_i)$ units of time if it were required for further computation. For convenience, we defined the following.

Residency or **currency** of an instance of an observable (of a data) is the amount of time it must be stored for future reference. Currency of a data θ obtained at a time t was its storage duration.

All our local variables could be partitioned into the following sets based on their individual systemic contributions, which in turn predicated their residency times. These sets were as follows.

2.3.7.1 *T(TRANSITIONAL)*

The variables in this set were derived and used (reported). A raw variable was not kept in the store after one obtained its statistics. For instance, if θ was such a variable then its expected value and variance $\mu_\theta = E(\theta)$ and $\sigma_\theta^2 = \text{var}(\theta)$ might be retained for a longer time in lieu of the variable θ . Let us assume a policy that retains minimally last n_θ such pair of values. These values could be aggregated (if aggregation is possible) to a higher level of abstraction yielding a model of type

$$\Omega_\theta : \mu_\theta \rightarrow c \quad \text{or} \quad \Sigma_\theta : \sigma_\theta \rightarrow d$$

Obviously, in such cases we would retain the functions Ω_θ and Σ_θ with longer currency.

2.3.7.2 *QT(QUASI-TRANSITIONAL)*

If $\theta \in QT$ then, its residency was determined in the following way. Let $f : \theta \rightarrow \zeta$ be a function, which aptly described some ζ as a function of θ . If the current θ was congruent to the function f , we did not store θ but, instead, stored the function f specification on the understanding that an inverse mapping would be able to restore θ (with some loss of precision) if we wanted to regain it. In this case, function form f would be stored for some time Δ_f .

If θ was incongruent to what the current accepted functional form f suggested, the latest value for θ would be stored. It was possible that such a sampled value might appear as a noise spike or it might be a genuine change in the value of θ . However, this could only be ascertained if a sequence of successive sampled values on θ showed a departure from the currently in use functional form f to some other form f' . Therefore, for data in QT set, we suggested the following.

Given $\theta_i = \theta(t_i)$ and the currently applicable functional form $f : \theta(t) \rightarrow \zeta$ for all $t_{last} \leq t < t_i$ excluding the sampled value θ_i on the ground that $f(\theta_i) \neq \zeta_i$, collate all successive sampled θ_i into another model given by $f' : \theta \rightarrow \xi$ until for some $\theta = \theta_n$. Thus, the currency of the variable θ was then the interval for which the old functional form was not acceptable and the new one was as yet unregistered, i.e.,

$$C(\theta) = t_n - t_i \quad \text{when} \quad t_i = \min(t) \quad \text{given that} \quad f(\theta(t)) \neq \zeta(t) \quad \text{and} \\ t_n = \min(t) \quad \text{given that} \quad f'(\theta(t)) \neq \xi(t)$$

In QT grouping, we would also have those variables entering for CUSUM analysis. Suppose that the variable θ was monitored at a node using its CUSUM statistics. At the beginning of a new model epoch we would monitor θ until the next model change point. Therefore, within that

epoch the only information of interest would be its expected value and its variance, but until these were obtained we would keep the raw θ values sampled and collected thus far. These might be just the last m set of raw θ values until a new model emerged.

An entity might require p sets of states to provide a measure. For instance, a link was defined between two nodes and, therefore, any performance measure of a link would require two state values. The currency of a link state was then at least as large as the minimum of the two state currencies. In this way, we could define the currency of a p -state entity (one needed at least p states from p distinct nodes comprising the entity) as

$$C(\vartheta | p - \text{states dependency}) = \min_j C_j(\theta)$$

The frequency of information exchange between regional nodes and the super-node would be predicated by this measure. The currency measure could not be too large lest it lose its relevance in the face of real-time characteristic of the observable domain.

2.3.8 SCALABILITY ISSUES OF TRAFFIC MODELS

We continued looking at scalability issues of traffic models, especially our own vector AR models. We were trying to determine how we could quantify network traffic as it related to a subnet or a group of nodes on the network instead of a single network device. For example, what if we wanted to know the utilization of a group of routers, but not switches? We currently have the ability to model an individual node based on multiple variables. In other words the AR model can be based on, say the traffic in, the traffic out, and the number of errors. One way of scaling this is to take the columns of the coefficient matrices corresponding to multiple variables of the same node as corresponding to single variable multiple nodes. For example, instead of having traffic in, traffic out, and traffic errors for a single node, we could use traffic in from three separate nodes. This appears to provide a useful measure of network activity but has not yet been incorporated into the RTNM.

2.3.9 ALTERNATIVE FRAMEWORKS

As stated previously, we experimented with various SNMP frameworks such as AdventNet, JMAPI, and JDMK/JMX. We decided to use the JDMK. The JDMK is a commercially supported management framework, which provides the most reliable SNMP management we have found so far.

JDMK also provides a framework for using agents and management beans or "Mbeans" to dynamically alter the actions an agent performs, or the way the agent performs. JDMK also allows agent to agent communication. These abilities of JDMK may open the door for a much more distributed and extensible management system.

We looked at several open-source network management projects, including GHOST and OpenNMS. None of these used a mathematical modeling approach for extracting information from the SNMP time series data that they collected and therefore, were not used.

2.4 TASK 4 – SOFTWARE SYSTEM IMPLEMENTATION

2.4.1 SOFTWARE EVALUATIONS

The process of software development between Phase I and Phase II of this SBIR varied largely in the fact that no software design tools or integrated development environment (IDE) were used. Therefore we approached the development by evaluating Java IDEs and software design tools. We selected IBM Visual Age for Java based on reviews about its features including the debugger. This phase II project was much larger than the phase I proof of concept; we knew we needed the features that IBM Visual Age for Java contained. Table 9 shows the comparisons among the software design tools.

Table 9. Comparison of Software Design Tools

Vendor	Product	Version	Platform	Description	Reqmnts	Pros	Cons	Cost	Comment
AdventNet	SNMP Management Builder	3.1	All/Java	SNMP GUI building tools	Provide Java-based SNMP GUI development beans	Free; many pre-made beans and such	Buggy; beans are not fully portable to non-AdventNet development tools	Free (Evaluation)	
Object Domain Systems	ObjectDomain	2.5	All/Java	UML-based visual object-oriented design tool	Provide visual design and roundtrip engineering	Free (evaluation)/Java	No Java 1.2 support; limited capabilities in the evaluation version	Evaluation: Free Professional: \$1300	
Tigris	ArgoUML	0.7	All/Java	UML-based visual object-oriented design tool	Provide visual design and roundtrip engineering	Free (evaluation)/Java	No activity diagrams; Java 1.2 not fully supported	Free	
TogetherSoft	TogetherJ	3.0	All/Java	UML-based visual object-oriented design tool for C++ and Java	Provide visual design and roundtrip engineering	Designed by Peter Coad, Very complete	Not widely used or supported	\$8000	Cost more than Rational Rose and not as prolific
Rational	Rational Rose	98i	NT	UML-based visual object-oriented design tool	Provide visual design and roundtrip engineering	Well tested and widely used	No Java 1.2 support or activity diagrams	\$4200	Currently using Rose2000 evaluation

2.4.2 SNMP DATA COLLECTION IMPLEMENTATION

An efficient data collection class for collecting SNMP data from multiple nodes was imperative to this project. Without collected data, utilization could not be projected. This class allowed us to query nodes with precision to gather information about nodes that we were querying. We focused our efforts on routers and switches. These nodes offered us the most traffic as well as the most congested areas. We could expand our study to other nodes fairly easily once we identified models as well as procedures for selecting models.

The information collected included MIB II variables from the physical layer up to the network layer. We focused on the IF or interface layer and IP or network layer. The data collected from the IF layer is listed below.

- ☐ ifInOctets (ifEntry 10) Description--Total number of octets in the interface for inbound traffic, including the framing characters.
- ☐ ifInUcastPkts (ifEntry 11) Description--Total number of subnetwork unicast packets delivered to a higher level protocol.
- ☐ ifInNUcastPkts (ifEntry 12) Description--Total number of broadcast/multicast subnetwork packets delivered to a higher level protocol.
- ☐ ifInDiscards (ifEntry 13) Description--Total number of inbound packets discarded due to lack of buffer space.
- ☐ ifInErrors (ifEntry 14) Description--Total number of inbound packets discarded due to errors.
- ☐ ifInUnknownProtos (ifEntry 15) Description--Total number of inbound packets discarded due to unknown protocol specification.
- ☐ ifOutOctets (ifEntry 16) Description--Total number of octets transmitted to the network from this interface (including framing characters).
- ☐ ifOutUcastPkts (ifEntry 17) Description--Same as for inbound traffic but now with changed direction.
- ☐ ifOutNUcastPkts (ifEntry 18) Description--Same as for inbound traffic but now with changed direction.
- ☐ ifOutDiscards (ifEntry 19) Description--Same as for inbound traffic but now with changed direction.
- ☐ ifOutErrors (ifEntry 20) Description--Same as for inbound traffic but now with changed direction.

- ❑ ifOutQLen (ifEntry 21) Description--The instantaneous length of the output packet queue in packets.

We also collected information on the TCP and UDP stacks. All this information could be stored in a database via JDBC.

Using JDBC API for MySQL, we linked our collection routine to a database. We used Sybase in Phase I because it was available at BAE SYSTEMS and had been used in the Phase I prototype. We extended to MySQL because it was freeware and had support for JDBC. The freeware DB allowed us to have multiple test networks without extra cost.

In keeping with Java's "write once, run anywhere" theme, we ran our application on multiple platforms--Sun's JDK on NT 4.0, Solaris 7, and Windows 95.

The network discovery algorithm was optimized. Aside from these changes, there were corrections to the algorithm. While the network discovery did seem to work effectively, we discovered that some of the SNMP queries sent to the Asynchronous Transfer Mode (ATM) switches were not being returned. In fact, we discovered that there was no detection when the queries were not being returned. This was very important. With the changes we made, we solved this problem and achieved an extremely complete network discovery. On one network, for example, we were able to detect almost 1,800 machines, yet only 124 of them were SNMP-capable. This was without using Ping or Traceroute to query an entire block of IP addresses as most discoveries do.

2.4.3 CUSUM AND AR DESIGN

The CUSUM filter Java implementation went smoothly based on our earlier Matlab implementation. We paid particular attention to the interface of these filters to other components of our software. Specifically, we expected the filters to be called in real time, and thus pass a reasonable slice of traffic measurements from data collectors.

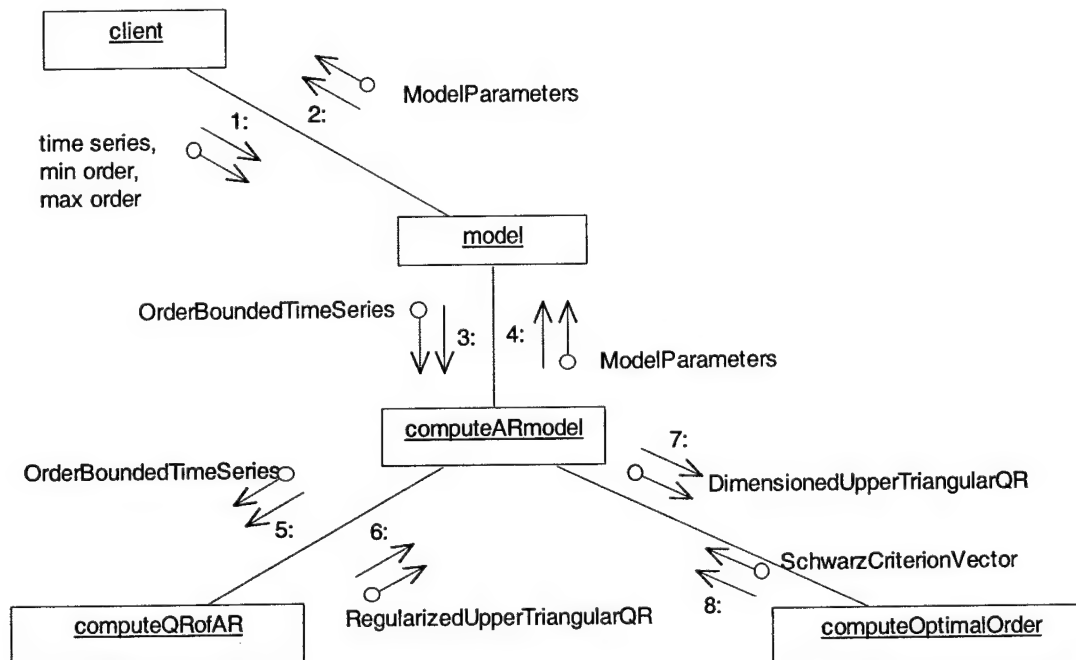
Concerning the data structures, the main issue in implementing the object model was the determination of the optimal data run length of traffic time series that achieves an acceptable real-time calculation of model change points, without unduly taxing traffic data collection and archive. From our experimental work, a length of about ten data points appeared acceptable. For analysis purposes, it was also decided to record the collection time for traffic data. Time was stored as String objects. The only constructor of the *TimeSeriesData* class of the object model was then expected to pass an array of not less than ten strings, and an array of not less than ten double-precision floating point numbers. Another object class that encapsulated both data and corresponding times could have been designed. However, after analysis, we reasoned that the design elegance did not pay for the time needed to pack and unpack objects of this encapsulating class.

Regarding the *CusumChangeData* class, two Boolean flags were used to signal a client whether the object it received from the CUSUM filter contained a new mean and/or variance. This

approach was necessary as the filter was normally called with the same number of data points, which might or might not produce a model change. If changes occurred, their time labels were returned as well. The *CusumFilter* class accumulated data and corresponding times that it received from clients. We used a length of a thousand and never exceeded it. The only constructor of *CusumFilter* took in initial data, which were used to compute initial values for mean, variance, and acceptable shifts for mean and variance. The actual algorithms used by the *filter()* method of the *CusumFilter* class were given in Section 2.4.3.

Exhibit 20 shows a somewhat abused UML collaboration diagram for the vector AR modeler. Normally, such a collaboration diagram is meant to capture the interaction between objects of an UML model. Here, we used it to show the interaction between the static methods of *MultivariateAutoregressiveModel*, which placed the various classes used in the AR modeler in a better perspective.

Exhibit 20. UML Collaboration Diagram for AR Modeler



Note that, with the numbering scheme of Exhibit 20, the correct sequence of calls from the initial client call until its return was 1-3-5-6-7-8-4-2. The *OrderBoundedTimeSeries* class was designed to allow the passing of the time series data as a Jampack Zmat object, and the range of putative model orders, as explained in Section 2.3.1.

The *RegularizedUpperTriangularQR* class held the upper triangular matrix, i.e., the R part, resulting from the QR decomposition of a matrix, called *regularizedKmatrix*, obtained from the time series data *Kmatrix* regularized by a regularization vector *regVec*. This latter was

essentially the square root of the *machine epsilon* times the vector computed from summing each column of the entry-wise absolute value of *Kmatrix*.

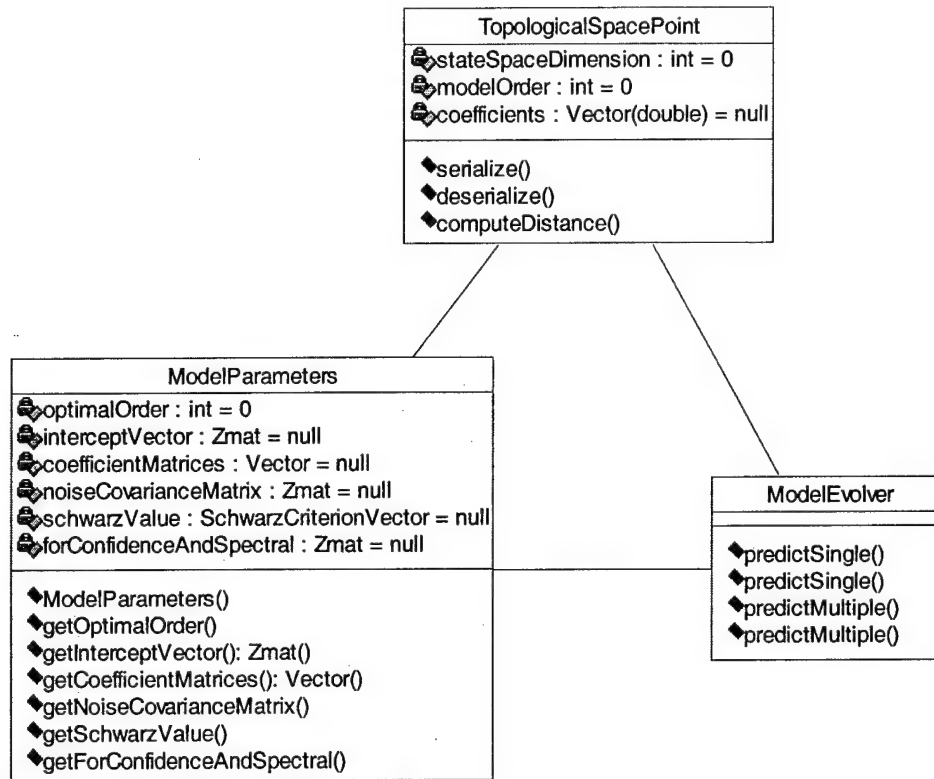
The *DimensionedUpperTriangularQR* class carried an upper triangular matrix (supposedly from QR decomposition), along with the range of putative model orders. In addition, it carried two integers for the state space dimension (i.e., the number of columns in the traffic time series) and a number of block equations (essentially, the number of time steps in the series minus the maximum order).

The *SchwarzCriterionVector* class held three *Zmat* objects, the first, called *schwarzValue*, being the only one of relevance to the current implementation. It contained a list of values for a given order selection criterion, for all the orders in the putative list. Although the variable name referred to the well-known Schwarz criterion, other criteria were also used in our implementation, including the Akaike criterion, and a modified Schwarz criterion. The other components of the *SchwarzCriterionVector* class might find use in later implementations, i.e., in spectral analysis.

The *ModelParameters* class defined what makes up a vector model. It contained an integer for the model order (the optimal one in a least-squares sense), a *Zmat* object for the intercept vector, a Java Vector of *Zmat* objects for the model coefficient matrices, and another *Zmat* object for the noise covariance matrix. For possible subsequent extensions, we also decided to include a *SchwarzCriterionVector* object, as defined above, and another *Zmat* object which might be useful in computing confidence intervals and spectral information.

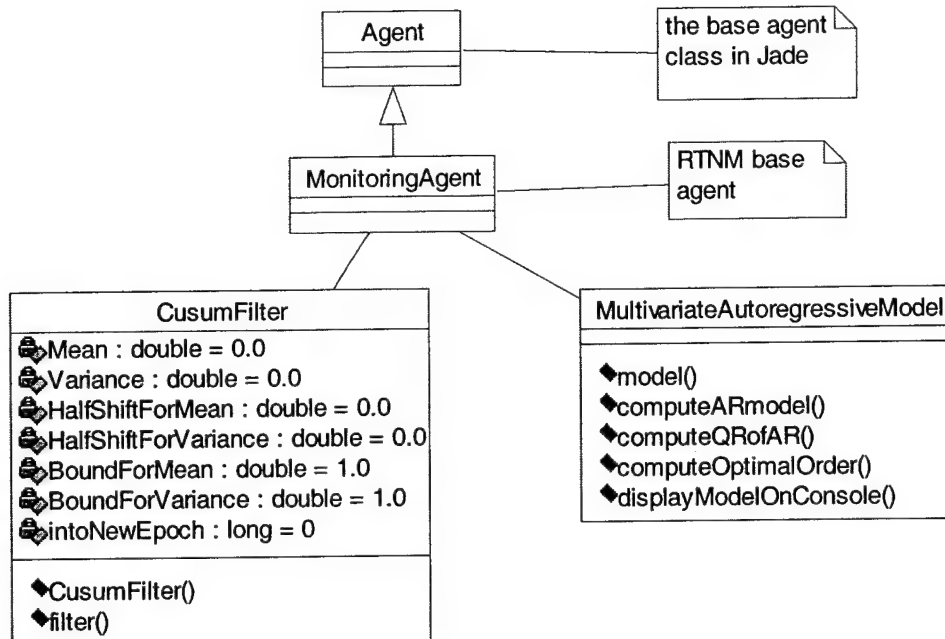
Note that the *MultivariateAutoregressiveModel* class did not contain any public data. The actual algorithms used by the vector AR modeler are given in Section 2.4.4.

For the topological operating space O_E , two Java classes were designed that related to the *ModelParameters* class according to Exhibit 21. The *TopologicalSpacePoint* class was the serialized version of the earlier *ModelParameters* class. In theory, we could have added additional attributes and methods to *ModelParameters* to handle the serialization chores; however, we gained speed efficiency in incorporating the new class, at a small cost of space. *TopologicalSpacePoint* stored both the state space dimension and the model order, as well as the actual array of doubles resulting from the serialization scheme explained in Section 2.3.5. The methods shown in Exhibit 21 are self-explanatory.

Exhibit 21. Class Diagram for Topological Space O_E and Model Evolver

The *ModelEvolver* class takes care of advancing a model in time, that is, to predict values of vector X_t in the notation of equation (28) of Section 2.3.5, at a single time point t or at multiple time points $t[]$. Such predictive values are useful in monitoring the life of a managed entity, and as a means for detecting pre-fault behavior.

The CUSUM filter and vector AR modeler fit within the agent framework discussed in section 2.3.6 and shown in Exhibit 22. If Jade-based research and development is pursued, then our agent class would inherit basic agent behaviors from the Jade-supplied *Agent* class. In addition to this functionality, we would also like to incorporate a rule-based engine, such as Jess (the Java Expert System Shell, from Sandia Labs), the Java incarnation of the CLIPS system. We also learned that the Jade agent system interfaced with Jess, which might simplify some design decisions if Jade were to be incorporated into our software. In our investigation, a rule corresponded to a pattern of events and statistic values that constituted a network problem. At that point, we were trying to peruse the outputs of our CUSUM filters.

Exhibit 22. Class Diagram for Monitoring Agents**2.4.4 THE IMPLEMENTATION OF THE CUSUM FILTER AND AR MODEL**

The algorithm used by the CUSUM filter proceeded as follows. First, the traffic data and corresponding times in the *TimeSeriesData* object passed to the *filter()* method were appended to what the *CusumFilter* object had already accumulated (recall that a filter object always returned after each change point, even if its data had not been completely processed). Next, we remembered how deep we were in an epoch, defined as a time interval without change. Then, we looped through the time series, each iteration performing the following tasks:

- ☐ Recomputing two-sided CUSUMs for the mean
- ☐ Recomputing two-sided CUSUMs for the variance
- ☐ Deciding if there were a change point for the mean
- ☐ Deciding if there were a change point for the variance.

Finally, we made the *CusumChangeData* object to be returned, making sure to update the data held by the *CusumFilter* object and the residual length.

Let μ and δ be the current mean value, and the permitted half shift from the current mean without triggering a signal. Let $\lambda(\mu, U)$ and $\lambda(\mu, L)$ be the last mean upper CUSUM and last mean lower

CUSUM computed. If the current data value was $d[i]$, then two temporary values τ_1 and τ_2 were obtained as follows :

$$\begin{aligned}\tau_1 &= \lambda(\mu, U) + d[i] - \mu - \delta \\ \tau_2 &= \lambda(\mu, L) - d[i] + \mu - \delta\end{aligned}$$

If $\tau_1 > 0$, then indeed $\lambda(\mu, U)$ was changed to τ_1 , and a running count of how many times the upper mean CUSUM had been non-zero was updated; a similar analysis was carried out for $\lambda(\mu, L)$ and associated quantities. The two-sided CUSUMs for the variance were obtained in a manner similar to that used for the mean, with the understanding that temporary CUSUM values for the variance were computed before any update to the mean was actually carried out.

The decision for the occurrence of a change point for the mean and the switch to a new mean value was based on a comparison between $\lambda(\mu, U)$ and the accepted bound for the mean. If such a decision were made, then all monitors were reset as well and appropriate new values for mean half shift and bound were computed. The change point decision for the variance was made in a similar fashion. In addition, a number of ancillary quantities, such as the residual length and the data accumulated by the filter, and their time labels, had to be updated. Testing of the CUSUM filter was done using both synthetic and real traffic data.

The algorithms used by the vector AR modeler proceeded as follows. The top level method, *model()* in Exhibit 20, simply made an *OrderBoundedTimeSeries* object from the raw traffic time series slice and the putative order range, then called the *computeARmodel()* method, produced the corresponding *ModelParameters* object. The *computeARmodel()* method itself started off by computing a *RegularizedUpperTriangularQR* object from the *OrderBoundedTimeSeries* object passed to it by calling the *computeQRofAR()* method, evaluating a QR factorization at the highest order specified. An order selection criterion vector, returned in a *SchwarzCriterionVector* object, was then computed via the *computeOptimalOrder()* method. A search was performed to find the minimal entry of this selection criterion vector. An optimal order p and an optimal parameter size m were next computed with the minimum order specified in the call being taken into account this time.

The next phase of the algorithm computed the intercept vector W and the coefficient matrices A_i . First, the R matrix from the QR decomposition was divided into relevant subblocks. Second, an augmented coefficient matrix was obtained from a regularization vector that was a byproduct of the QR decomposition and from solving a linear system of equations. Next, the noise covariance matrix was computed from scaling an appropriate subblock of R . Finally, a few other matrices, relevant for an eventual computation of spectral information and confidence intervals, were computed.

The *computeQRofAR()* method was almost like a classical QR decomposition except that it was not performed on the original matrix resulting from the traffic time series. Instead, predictors and regulators were injected into the matrix. The *Zqrd* suite in Jampack was used to perform the actual QR decomposition.

The *computeOptimalOrder()* method had the upper triangular matrix *R* of the QR decomposition passed to it. Then it set up a collection of vectors, including the order selection criterion vector and the noise covariance determinant logarithm vector, of maximum length $max - min + 1$, where *max* and *min* denoted the putative largest and smallest model orders. The noise covariance matrix was computed as the inverse of a subblock of *R*. The essential part of *computeOptimalOrder()* was, however, a backward iteration through the range of possible orders, where the noise covariance inverse was updated through Cholesky decompositions and left divisions. Here, we noted that we found the Jampack to be lacking the precision we required for Cholesky decompositions. We then switched to the JAMA package for these decompositions, then went back to the Jampack to update the determinant logarithm vector and the order selection vector. Testing of the vector AR modeler was done using both synthetic and real traffic data.

Algorithms for the *TopologicalSpacePoint* class, at this point in time, were confined to serialization and deserialization of vector AR models, and to the computation of the topological distance between any two objects of the class. The natural order imposed by equation (28) in Section 2.3.5 and the structure of the *ModelParameters* class formed the basis of the serialization. The topological distance was computed via the Froebenius norm of a Zmat object in the Jampack, although it was rather straightforward to compute other equivalent norms directly.

Algorithms for monitoring agents were, at that time in the project, at a very early stage in their development. An agent was a Java serializable thread object whose capabilities were stored as hashmaps of languages and ontologies. It might be interesting to design a new language for communication between network manager agents and managed device agents; however, the languages provided by the FIPA specifications were probably sufficient, if enhanced by the needs of SNMP. A more worthwhile task lay in connecting both manager and managed agents to the rule-based engine provided by Jess. In particular, a translation between FIPA message structures and Jess facts should be made, so that rule bases written as CLIPS files might be executed by an agent.

2.4.5 EVENT CORRELATION ENGINE

Work on the event correlation engine and diagnosis subsystems was begun. Specifically we looked at the SMARTS system, which is System Management ARTS Incorporated's object-oriented diagnostic modeling and cookbook correlation. Instead of using solely a rule-based system, SMARTS uses what they call a *cookbook*. The cookbook, which is based on an object-oriented diagnostic model (OODM), is an extremely efficient mechanism that addresses some of the limitations of rule-based systems. Again, given our time constraints, we did not pursue this, and decided on the following.

The RTNM's diagnostic capabilities would be derived from a limited set of variables. These variables, when combined with a Bayesian network and rule-based system would provide a proactive management system capable of not only detecting network problems before they became failures, but would provide information as to the cause of these problems. The variables

used in the diagnosis subsystem consisted of the messages (events) being generated by the various RTNM components. These events consisted namely of Java events, SNMP events, and traffic events.

Java events were the error or status messages generated from within the Java language. Java errors might occur when there were network problems, database problems, or even internal communications problems among the various software modules.

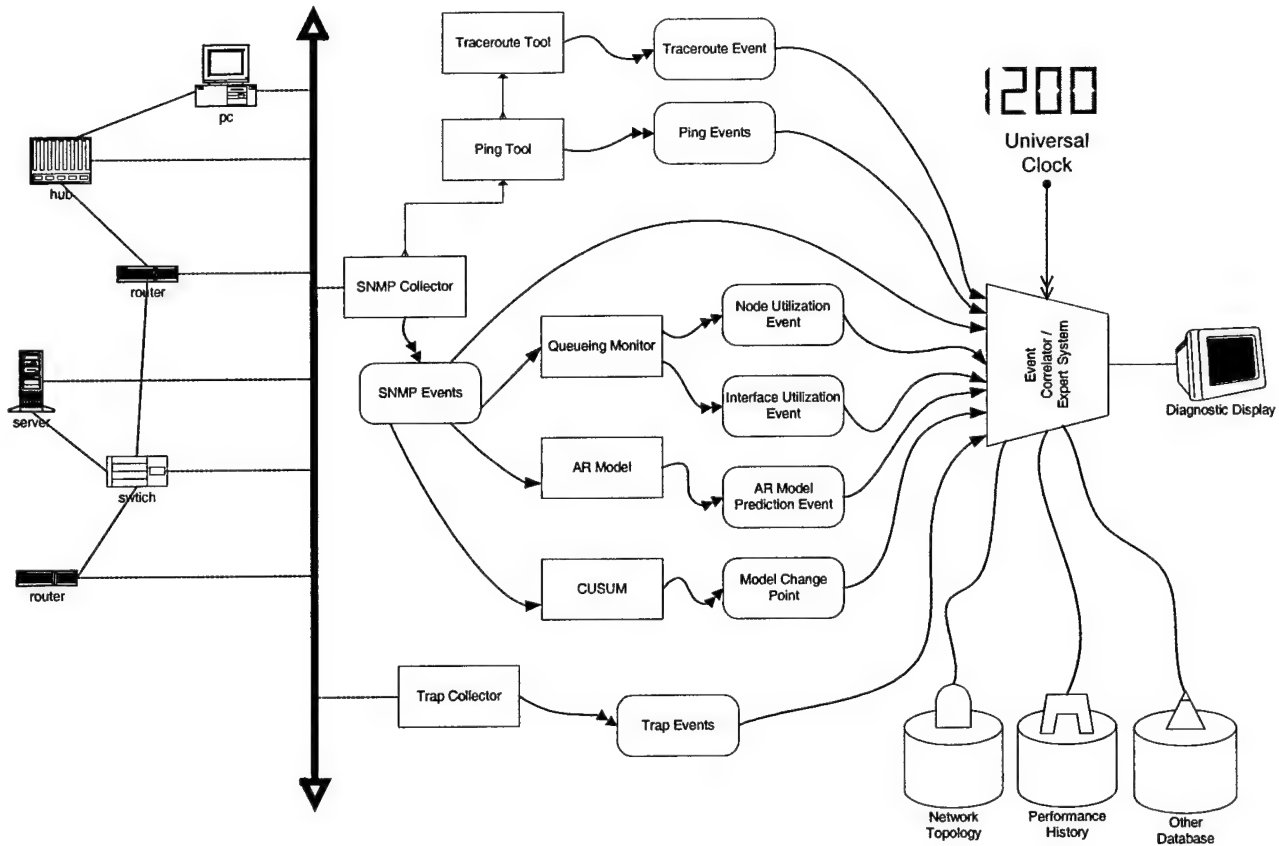
SNMP events were messages relating to the monitoring of the network traffic and devices. These events might be in response to active SNMP queries or as a result of network devices experiencing a problem. The events generated by a device were known as *traps*.

Traffic events occurred when there were changes in traffic patterns, high traffic loads, or anomalous traffic activity. These traffic events would be "fired" when one or more of the modeling subsystems detected a significant problem or change in the network traffic.

The Java implementation of the diagnostic subsystems would use the following tools.

- ☐ Rule-based system and fuzzy logic rules: Jess and FuzzyJ add-on.
- ☐ Bayesian belief networks: JavaBayes package.

The overall event diagnosis / correlation architecture for the RTNM is shown in Exhibit 23.

Exhibit 23. Event Correlation Architecture**Event Correlation Architecture For RTNM****2.4.6 JAVA OPTIMIZATION**

Now that the entire architecture of the application was complete, we were able to go back and optimize some of our algorithms. Because Java is notoriously slow and memory hungry, it was imperative to optimize wherever we could. The initial results were impressive. Much of the RTNM initialization and Graphical User Interface (GUI) response times were halved. The network discovery tool, while still not perfect, appeared to be almost four times faster and required much less of the system's resources.

The final look and layout of the RTNM interface was established. The incorporation of the various RTNM components into one central interface was achieved (see Exhibit 24). The RTNM now had an uncluttered top level view of the management tools. These tools included network discovery, utilization, prediction, and monitoring for both the Internet Protocol (IP) and Interface layer as well access to Ping, Traceroute, trap browser, and database configuration tools. Each of these had its unique drill-down capabilities as shown in Exhibits 25, 26, and 27.

Exhibit 24. RTNM Portal

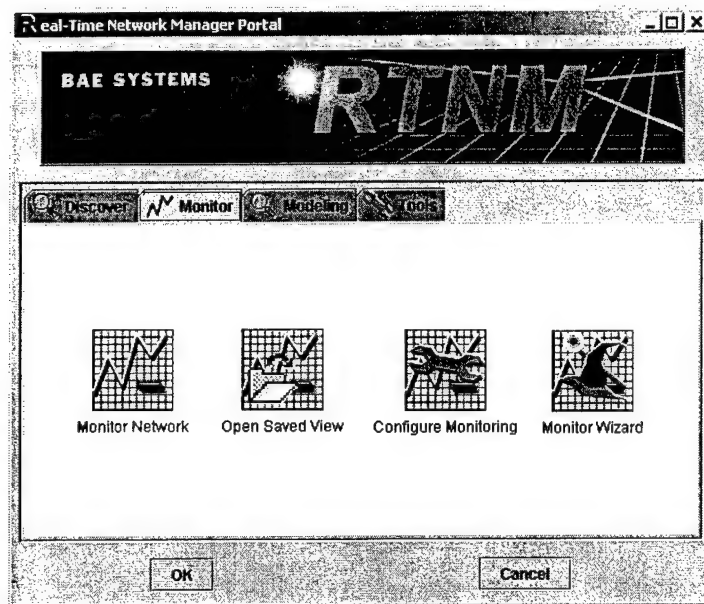


Exhibit 25. RTNM Network View

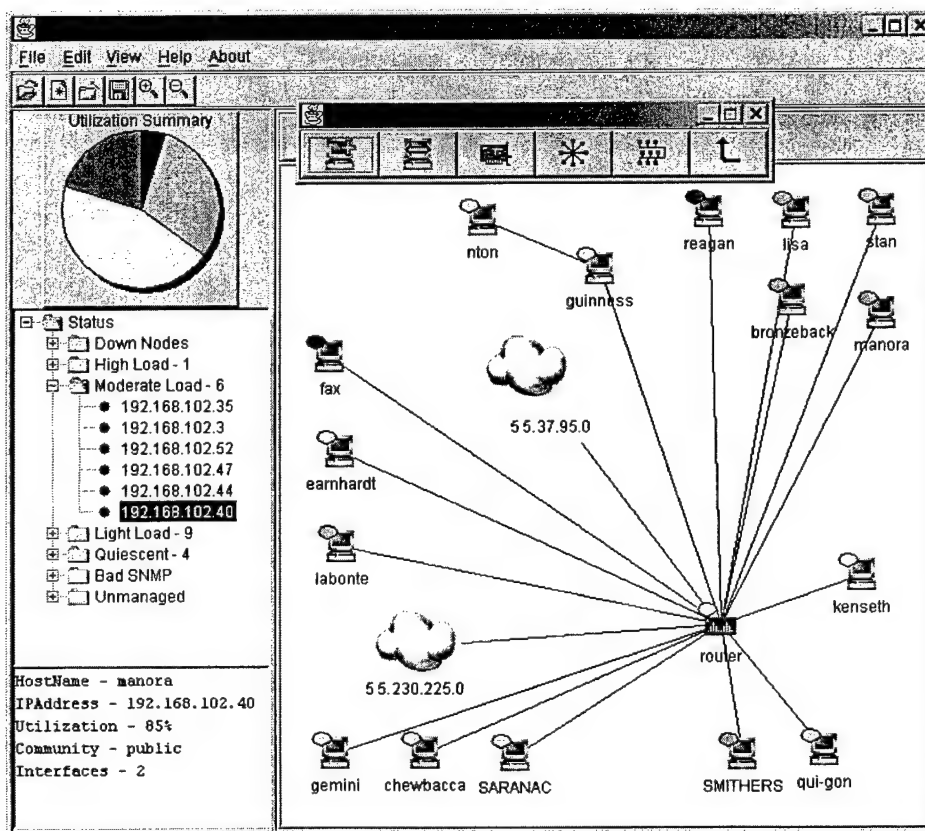


Exhibit 26. RTNM Node IP View

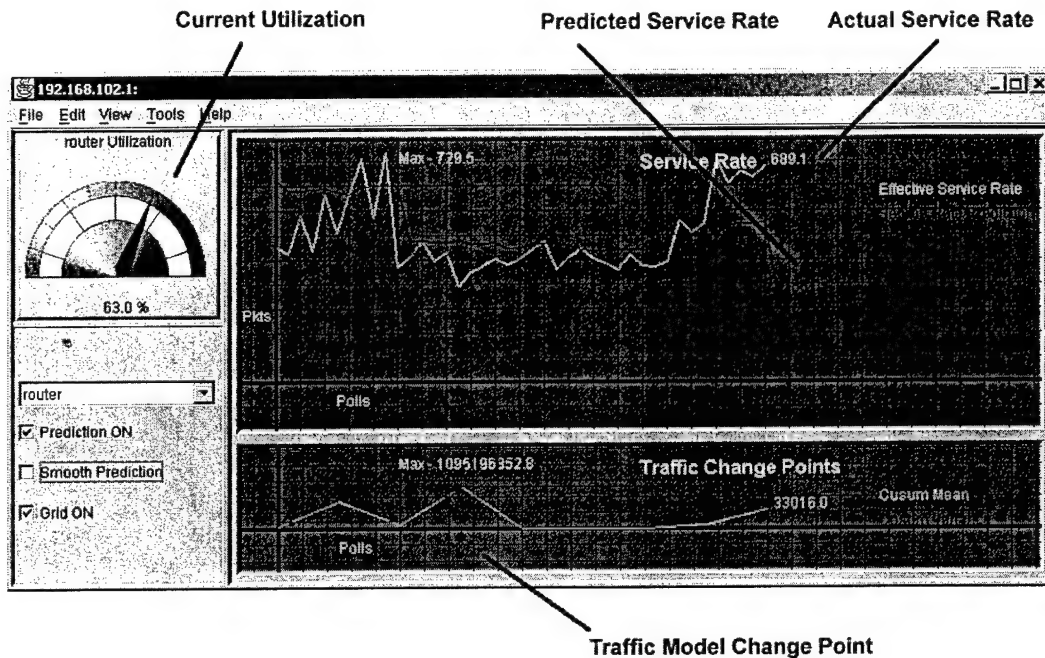


Exhibit 27. RTNM Tools

Field	Type	Null	Key	Defa
ipAddress	varchar(15)		PRI	
hostName	varchar(30)	YES		
pubComm...	varchar(30)	YES		
privComm...	varchar(30)	YES		
isAlive	char(1)	YES		
isSNMP	char(1)	YES		
discoverTi...	timestamp	YES		

ipAddress	hostName	o	privCommu...	isAlive	isSNMP	discoverTime	sysObjectID	ipForwarding	maxRate	comm
192.168.1...	kenseth	...		Y	Y	2001-06-14 ...	1.3.6.1.4.1.311.1.1.3.1.1	N	47.3333	
192.168.1...	stan	...		Y	Y	2001-06-14 ...	1.3.6.1.4.1.311.1.1.3.1.1	N	13.3333	
192.168.1...	lisa	...		Y	Y	2001-06-12 ...	1.3.6.1.4.1.2021.250.3	N	53.7249	
192.168.1...	reagan	...		Y	Y	2001-06-14 ...	1.3.6.1.4.1.311.1.1.3.1.3	N	10576.07...	
192.168.1...	robinson	...		Y	Y	2001-06-12 ...	1.3.6.1.4.1.311.1.1.3.1.3	Y		
192.168.1...	router	...		Y	Y	2001-06-14 ...	1.3.6.1.4.1.9.1.17	Y	827.5632	
192.168.1...	guinness	...		Y	Y	2001-06-14 ...	1.3.6.1.4.1.311.1.1.3.1.1	N	420.0	
192.168.1...	nton	...		Y	Y	2001-06-14 ...	1.3.6.1.4.1.311.1.1.3.1.1	N	36.0	
192.168.1...	fax	...		Y	Y	2001-06-14 ...	1.3.6.1.4.1.311.1.1.3.1.1	N	719.7071	
192.168.1...	earnhardt	...		Y	Y	2001-06-14 ...	1.3.6.1.4.1.311.1.1.3.1.1	N	79.0885	
192.168.1...	labonte	...		Y	Y	2001-06-14 ...	1.3.6.1.4.1.311.1.1.3.1.2	N	16.6113	
192.168.1...	gemini	...		Y	Y	2001-06-14 ...	1.3.6.1.4.1.311.1.1.3.1.1	N	24.6667	
192.168.1...	chewbacca	...		Y	Y	2001-06-12 ...	1.3.6.1.4.1.311.1.1.3.1.1	N	24.4224	
192.168.1...	SARANAC	...		Y	Y	2001-06-14 ...	1.3.6.1.4.1.311.1.1.3.1.1	N	319.6501	

2.4.7 TRAFFIC GENERATION TOOLS

In order to provide an adequate demonstration of the capabilities of the RTNM, it was necessary to research traffic generation applications. These would allow us to create traffic on a network and provide better test and demonstration scenarios. Outlined below are five free traffic generators that we tested with our application.

2.4.7.1 *NETPERF*

An employee at Hewlett-Packard (HP) designed NetPerf. HP offers no protection to the user should something go wrong; however it is (seemingly) an effective tool. NetPerf is a benchmark that can be used to measure the performance of many different types of networking. It provided tests for both unidirectional throughput and end-to-end latency. The environments currently measurable by NetPerf include:

- ☐ TCP and UDP via BSD Sockets
- ☐ DLPI
- ☐ Fore ATM API
- ☐ HP HiPPI Link Level Access

See URL: <http://www.netperf.org/>

2.4.7.2 *PSC TRENO SERVER*

TReno is being developed at Carnegie Mellon University at the Pittsburgh Supercomputing Center (PSC). The program has several government grants for work on the project. TReno tests the network under a load similar to TCP and it uses the same technique as Traceroute. It is single-threaded and therefore the user must wait to test until there is no other test traffic on the network. TReno is run from the PSC server or by downloading the software.

See URL: [http://www.psc.edu/networking/TReno info.html](http://www.psc.edu/networking/TReno%20info.html)

2.4.7.3 *TTCP AND NTTCP*

This application was originally written to move files around but it became the classic throughput benchmark or load generator. With the addition of support for sourcing from /dev/null it has spawned many variants including support for DP, data pattern generation, page alignment, and even alignment offset control.nttcp that allows mcast UDP transfers.

See URLs: ttcp is located at <ftp://ftp.arl.mil/pub/ttcp/>
nttcp is located at <http://www.leo.org/~elmar/nttcp/>

2.4.7.4 NETPIPE

NetPipe, written in Java, is a protocol-independent performance tool. It has many of the same qualities of ttcp and NetPerf.

See URL: <http://www.scl.aineslab.gov/netpipe/>

2.4.7.5 CHARIOT

This program also generates traffic to test a LAN. It supports multiple protocols but it does not seem to be as robust as some others found. Chariot tests the performance of networked applications, stress tests network devices, predicts application performance prior to deployment, and troubleshoots network performance problems. You can use Chariot's performance data to optimize your network and measure the performance impact of proposed network changes.

See URL: <http://www.netiq.com/Products/Network Performance/Chariot/Default.asp>

2.4.8 SOFTWARE TEST SITES

In order for us to test our applications we used four sites with different topologies and networks for testing and collecting data. The first network on which we performed collection was at BAE SYSTEMS Rome Operations. This is a small, 25-node network connected to the Internet. It is not heavily used and is bursty in nature.

The second was the network at the Air Force Research Laboratory – Rome Research Site. This network is a large, 1600+ node network with many subnets and multiple ELANs and VLANs. Utilization is medium to heavy and it contains many different types of SNMP nodes including Fore ATM switches, Fore powerhubs, hosts, printers, and CD-ROM servers. Some of the powerhubs contain over 100 ports, which is a network management challenge in itself.

The third site was a university in the northeastern United States. This university gave us permission to monitor their network, provided that we did not publicize their identity or our findings. This is also a large network with many on-line systems. The SNMP nodes include Bay Networks Centillion ATM switches to HP bridges and a Cisco router. This network yielded some interesting findings, especially since students used the network around the clock.

The fourth site was at the Air Force Research Laboratory – Rome Research Site. It was in the Joint Integration and Test Facility on a classified network. We wanted to see how RTNM would react to a very different and secure network.

Each test site provided a uniqueness that stressed RTNM in many ways. Some helped us fix problems in RTNM; others provided ways to improve RTNM. All in all it was excellent having different test environments.

3.0 APPENDIX

3.1 RTNM INSTALLATION INSTRUCTIONS

3.1.1 INTRODUCTION

This document briefly describes what is needed for installation and use of RTNM on your network.

3.1.2 SOFTWARE REQUIRED

The software required for RTNM includes:

- ☐ JDK 1.3
- ☐ MySQL version 3.2X
- ☐ RTNM software which includes APIs such as JDMK 4.2, JAMA, JAMPACK, and MM MySQL JDBC version 0.9

A version of UCD-SNMP for the PC is also included for testing purposes.

3.1.3 SOFTWARE INSTALLATION AND CONFIGURATION

Software installation includes MySQL, JDK, and RTNM software.

3.1.3.1 MYSQL INSTALLATION AND CONFIGURATION

MySQL is the first application that needs to be installed. Run the Setup.exe file found in the MySQL directory on the CD. Follow the MySQL installation instructions. Once MySQL is running, you need to create a database, users, and give access to RTNM.

```
mysql -u root -p mysql
```

Enter password:

Reading table information for completion of table and column names

You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 7 to server version: 3.22.27

Type 'help' for help.

```
mysql> create database snmp ;
Query OK, 1 row affected (0.00 sec)
```

```
mysql> insert into user (Host,User>Password) values('localhost','rtnm',PASSWORD(
'rtnm11'));
Query OK, 1 row affected (0.00 sec)
```

```
mysql> flush privileges ;
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> grant all on snmp.* to rtnm ;
Query OK, 0 rows affected (0.01 sec)
```

3.1.3.2 JDK 1.3 INSTALLATION AND CONFIGURATION

Double click the icon : j2sdk-1_3_1-win and follow the instructions.

3.1.3.3 RTNM INSTALLATION AND CONFIGURATION

The installation of RTNM will use the jar command that will verify that your path is set up correctly. First of all you need to identify a directory on which RTNM will be installed, i.e., C:\RTNM. Next run the command `jar -xvf <drive letter of cdrom>:rtnm.jar` from the installation directory. This installs RTNM into the directory and is ready to run.

There is a run batch file and a run script for Windows and UNIX systems, respectively. The RTNM launch portal can be started with **"run"** on Windows systems or **"/run"** on UNIX systems.

Note: The run script can be made "executable" on UNIX systems by issuing the `chmod` command from the

RTNM directory:

`"chmod 755 run"` will allow you to just type **"run"**.

Next select discover to discover your network. Once the network is discovered, RTNM can proactively manage your network.

3.2 DATABASE MANAGEMENT SYSTEMS AND TOOLS EVALUATION MATRIX

Vendor	Product	Version	Platform	Description	Reqmnts	Pros	Cons	Cost	Comment
Sybase	SQL server	Unknown	Windows/ Solaris	Database management system	Provide network information storage and retrieval	Well supported	Cannot be redistributed	Expensive	Worked well but we could not legally install it on our test networks
Mysql	Mysql	3.22.77	All	Database management system	Provide network information storage and retrieval	Free and easily installed	Does not allow stored procedures; a limited number of JDBC drivers available	Free GNU	We are currently using this
Hughes Technologies	MSQL	Unknown	UNIX	"mini" Database management system	Provide network information storage and retrieval	Free	Not enough database features; no good JDBC drivers available	Free	
Sybase	Jconnect	Unknown	All/Java	Java JDBC driver for the Sybase database	Provide out database connectivity from Java	??	??	Evaluation	
MM Mysql	mm mysql JDBC driver	2.0.2	All/Java	Java JDBC driver for the Mysql database	Provide out database connectivity from Java	Best existing driver for MySql	No JDBC 2.0 support (yet)	Free	We are currently using this driver

3.3 MATH PACKAGE EVALUATION MATRIX

Vendor	Product	Version	Platform	Description	Reqmnts	Pros	Cons	Cost	Comment
NIST and MathWorks	JAMA	Beta	All/Java	Construction and manipulation of real, dense matrices	Provide matrix operations for AR Modeling	Understandable to non-experts		Free	
NIST and University of Maryland	JAMPACK	Beta	All/Java	A collection of classes designed to perform matrix computations in Java applications	Provide matrix operations for AR Modeling	Easier to use than JAMA	No determinants; all matrices are complex	Free	We are currently using this package
Operations Research	OR-Objects	1.2.4	All/Java	Java classes for scientific and engineering applications	Provide matrix operations for AR Modeling	Pros	Lacks decomposition operations	Freeware	
University of Wisconsin Madison	Octave	2.0.16	All	Matlab clone primarily intended for numerical computations	Provide matrix operations as efficiently as Matlab (for free)	Ability to use our existing Matlab ARFit scripts	Many external/platform specific libraries required	Free (GNU)	Installation was to complex for redistribution
MathWorks	Matlab	5.x	All	Numerical computation and visualization package	Provide numerical methods for network modeling	Excellent and efficient libraries matrix operations	Expensive and non-redistributable		This package has been used for the development of the network modeling

4.0 GLOSSARY

API	Application Programmer's Interface
AR	Autoregressive
ARMA	Autoregressive Moving Average
ATM	Asynchronous Transfer Mode
CDRL	Contract Data Requirements List
CORBA	Common Object Request Broker Architecture
CUSUM	Cumulative Sum
DARPA	Defense Advanced Research Projects Agency
DC	Direct Current
ELAN	Emulated Local Area Network
EMS	Equivalent Markovian Server
FFT	Fast Fourier Transform
FIPA	Foundation for Intelligent Physical Agents
FIPA-OS	FIPA Operating System
GUI	Graphical User Interface
IID	Independent Identically Distributed
IP	Internet Protocol
JAMA	Java Matrix Package
JDBC	Java Data Base Connectivity
JDMK	Java Dynamic Management Kit
Jess	Java Expert System Shell
JMAPI	Java Management Application Programmer's Interface
JMX	Java Management Extensions
MASIF	Mobil Agent Software
MESRA	Markov Equivalent Server's Rate Abstraction
MIB	Management Information Base
NIST	National Institute of Standards and Technology
OID	Object Identification Number
OMG	Object Management Group

OODM	Object-oriented Diagnostic Model
RMI	Remote Method Invocation
RMON	Remote Monitoring
RTNM	Real-Time Network Manager
SBIR	Small Business Innovative Research
SMARTS	System Management ARTS
SNMP	Simple Network Management Protocol
SUNY IT	State University of New York Institute of Technology
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UML	Unified Modeling Language
VLAN	Virtual Local Area Network